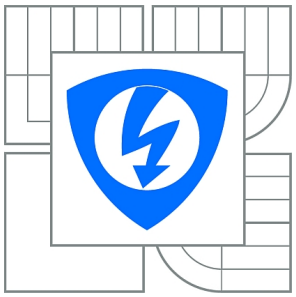
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

SMĚROVACÍ PROTOKOLY PRO MANET SÍTĚ SE ZAMĚŘENÍM NA QOS

ROUTING PROTOCOLS FOR MANET NETWORK WITH FOCUS ON QOS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

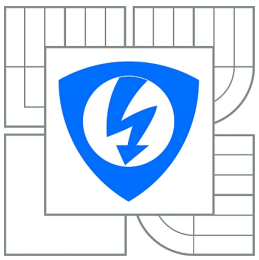
AUTOR PRÁCE — Bc. ATTILA POTFAY

AUTHOR

VEDOUCÍ PRÁCE — Ing. PAVEL VAJSAR

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Attila Potfay	***ID:***	106730
Ročník:	2	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Směrovací protokoly pro MANET sítě se zaměřením na QoS

POKYNY PRO VYPRACOVÁNÍ:

V rámci diplomové práce bude nutné seznámit se s MANET (Mobile ad hoc) sítěmi a simulačním prostředím ns3. Prostudovat a teoreticky zpracovat stávající možnosti řešení kvality služeb (QoS - Quality Of Services) u těchto sítí. V prostředí ns3 vytvořte model MANET sítě, který bude obsahovat podporu směrovacího protokolu dle zadání. Proveďte simulace, kterými porovnáte vliv podpory kvality služeb na klíčové parametry provozu. Dále oověřte možnosti zavedení reálných zařízení do procesu simulace v prostředí n3.

DOPORUČENÁ LITERATURA:

[1] ILYAS, M.: The Handbook of Ad Hoc Wireless Networks. Boca Raton: CRC Press, 2003, ISBN: 0-8493-1332-5.

[2] MOHAPATRA, P., KRISHNAMURTH, S.: Ad Hoc Networks: Technology and Protocols. Boston: Springer Press, 2005, ISBN: 0-387-22689-3.

Termín zadání:	11.2.2013	***Termín odevzdání:***	29.5.2013

Vedoucí práce: Ing. Pavel Vajsar
Konzultanti diplomové práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Hlavním úkolem této diplomové práce je simulace směrovacího protokolu AODV (Ad hoc On-Demand Distance Vector Routing) v prostředí Network Simulator ns-3, realizovat model MANET (Mobile Ad-hoc Network) sítě, který obsahuje podporu kvality služeb (QoS - Quality Of Service). Dále implementovat protokol AODV do reálných zařízení a zavedení těchto uzlů do procesu simulace pomocí simulátoru ns-3. Tato práce k tomu poskytuje teoretické základy: zabývá se základními vlastnostmi MANET sítí, podrobně popisuje směrovací protokoly MANET s podporou QoS, poskytuje informace o simulačním prostředí Network Simulator ns-3, podrobně popisuje stávající řešení implementace protokolu AODV do reálných zařízení a poskytuje informace o metodě zapojení reálných zařízení do simulace.

KLÍČOVÁ SLOVA

MANET, AODV, ns-3, QoS, FTP, VoIP, simulace, NetAnim, EmuNetDevice, TapNetDevice, AODV-UU, WinAODV

ABSTRACT

The main task of this master's thesis is to simulate the routing protocol AODV (Ad hoc On-Demand Distance Vector Routing) to the Network Simulator ns-3 environment, and to realize a model of MANET (Mobile Ad-hoc Network) network with the support of Quality Of Service (QoS). Further implement the protocol AODV in real devices and the involvement of such nodes in the simulation process using the simulator ns-3. This work provides the theoretical basics: it deals with the primary characteristics of MANET, describes in detail the routing protocols MANET with support of QoS, provides information about the Network Simulator ns-3 simulation environment, describes in detail the existing implementation solutions of the protocol AODV in to real devices and provides information about the methods of involvement real devices to the simulation.

KEYWORDS

MANET, AODV, ns-3, QoS, FTP, VoIP, simulation, NetAnim, EmuNetDevice, TapNetDevice, AODV-UU, WinAODV

POTFAY, Attila *Směrovací protokoly pro MANET sítě se zaměřením na QoS*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 80 s. Vedoucí práce byl Ing. Pavel Vajsar

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Směrovací protokoly pro MANET sítě se zaměřením na QoS“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Pavlu Vajsarovi za odborné vedení, cenné rady a připomínky. Dále bych chtěl poděkovat Ing. Petru Hoškovi za pomoc při gramatické korektuře této práce, své rodině, která mi poskytla podporu a dostatek času, a mé snoubence, za její trpělivost a motivaci.

Brno ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .
(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

V dnešní době již nejsou klasické Wi-fi sítě dostatečně mobilní, zejména kvůli nejrůznějších omezení. Jedním z omezení je například dosah přístupového bodu. Proto před pár lety vznikla nová technologie, která se touto problematikou zabývá a je nazývána Mobile Ad-hoc (MANET), což je nezávislé řešení bez použití infrastruktury.

MANET sítě používají různé směrovací protokoly, které jsou schopny zajistit funkčnost sítě i při mobilitě uzlů a stálých změnách topologie. Směrovací protokoly MANET mají spoustu vlastností, ale nejdůležitější ze všech je možnost podpory kvality služeb QoS. Veliká část práce je zaměřena na otestování a vyhodnocení právě této vlastnosti.

Prvním cílem této práce je seznámení se s MANET sítěmi a se stávajícími možnostmi řešení kvality služeb (QoS - Quality Of Services) u těchto sítí. Dále je třeba se seznámit se síťovým simulačním prostředím *Network Simulator ns-3* a popsat jeho vlastnosti. Dalším cílem je vytvořit model MANET sítě (v prostředí `ns-3`), v němž bude používán směrovací protokol AODV se zapnutou podporou kvality služeb.

Dalším cílem této práce bylo nastudovat a aplikovat možnosti reálné implementace směrovacího protokolu AODV a pokusit se vytvořit reálnou síť. Cílem bylo ještě nastudovat zavedení reálných zařízení do procesu simulace v simulační prostředí `ns-3` a pokusit se využít takové zařízení v simulaci.

Na závěr je třeba vyhodnotit dosažené výsledky z provedených simulací a z vytvořené sítě.

V první kapitole je uvedeno stručné seznámení se sítěmi MANET, jak fungují a jaké jsou hlavní typy protokolů.

V druhé kapitole je seznámení se stávajícími možnostmi řešení kvality služeb v MANET sítích. Zde jsou popsány specifické problémy MANET sítě z hlediska QoS. Dále jsou popsány nejdůležitější metriky a modely kvality služeb. Poslední podkapitola 2.3 této části práce se zabývá rozdělením směrovacích protokolů pro MANET s podporou QoS a zpracováním vlastností třech protokolů.

Třetí kapitola se věnuje simulačnímu programu *Network Simulator ns-3*. Je zpracován popis tohoto programu a jeho vlastnosti. Dále jsou zpracovány stávající metody zapojení reálných zařízení do simulace.

Ve čtvrté kapitole jsou popsané dostupné implementace směrovacího protokolu AODV do platformy JAVA a do operačního systému Windows a Linux. Pro operační systémy jsou zpracovány i postupy implementací v kapitole 6.

Páta kapitola práce se zabývá již nasimulovanou MANET sítí. Jsou uvedeny základní vlastnosti programu a podrobně popsána každá část zdrojového kódu. Na závěr této kapitoly jsou zhodnoceny výsledky simulace.

Sedmá kapitola je věnována dokumentaci vytvořené sítě postavené na protokolu AODV. Je prezentována topologie a funkčnost sítě pomocí směrovací tabulky a procházejících paketů mezi uzly.

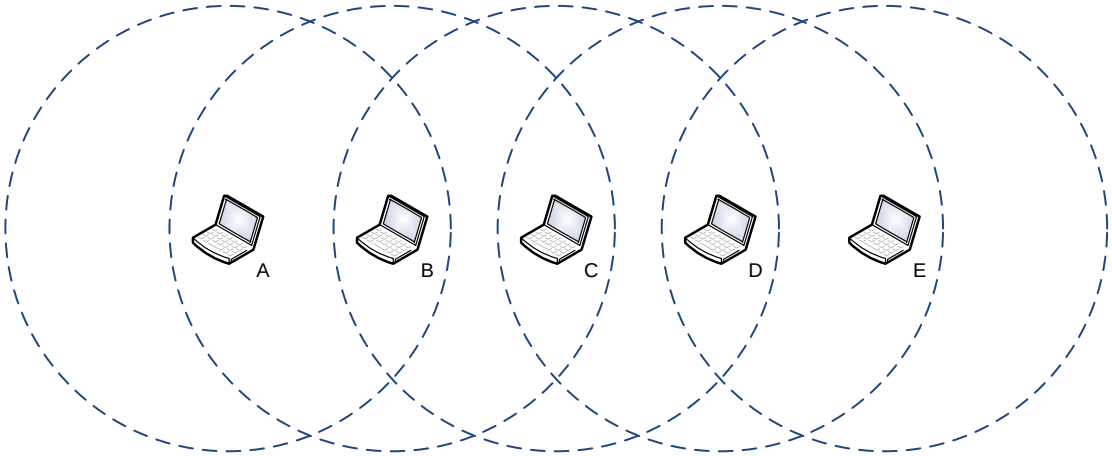
Osmá, poslední kapitola hlavní části práce, se zabývá zavedením reálných zařízení do procesu simulace. Dva uzly komunikují pomocí protokolu AODV a na obou uzlech běží simulace.

1 MANET SÍTĚ

Tato kapitola se bude zabývat MANET sítěmi, jejich hlavními vlastnostmi a typy protokolů.

1.1 Obecně o MANET sítích

MANET[1] (Mobile Ad-hoc Networks) jsou bezdrátové sítě, které fungují velmi stabilně i v případech, kdy dochází ke změnám topologie sítě v čase i v prostoru. Tyto sítě nepotřebují žádnou infrastrukturu nebo centralizovanou administrativu (např.: rozbočovače, směrovače). Vše je realizováno tak, že se uzly (*nodes*) v MANET sítích nechovají jen jako hosté, ale i jako směrovače. Každý uzel, který se chce stát součástí sítě, musí být schopen dosáhnout na ostatní uzly, které už v síti jsou. Jestliže uzel A dosáhne na uzel B, který je prvkem MANET sítě (což představují uzly B, C, D, E) a jsou schopny mezi sebou komunikovat, uzel A se stane součástí sítě. V případě, že uzel E je mimo pokrytí uzlu A a chtějí si mezi sebou vyměňovat data, tak přenos bude probíhat přes uzly B, C, D. [1]



Obr. 1.1: Ad-hoc síť

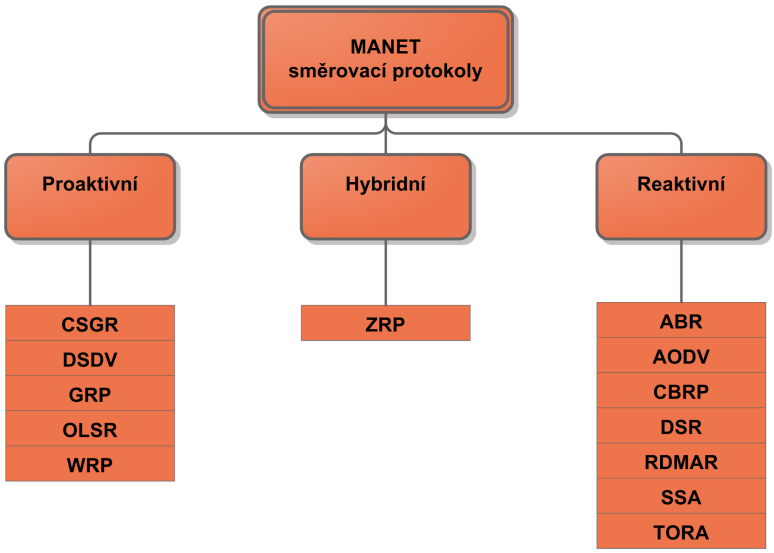
Existuje více řešení v problematice směrování v Ad-hoc sítích, tzv.: směrovací algoritmy a protokoly. Díky tomu, že uzly v MANET sítích nepotřebují žádná zařízení na to, aby jejich činnost byla organizovaná, jsou směrovací protokoly implementovány přímo do jednotlivých stanic.[1]

[1]Technologie MANET RFC 2501 je dostupné na http://www.ietf.org/rfc/rfc2501.txt

1.2 Hlavní typy protokolů pro Ad-hoc sítě

Směrovací protokoly v MANET sítích můžeme rozdělit na tři velké skupiny[1]:

- **Proaktivní protokoly** – udržují aktualizovaný seznam o cílech a o jejich trasách takovým způsobem, že pravidelně distribuují směrovací tabulky přes síť.
- **Reaktivní protokoly** – na požádání (on demand) najdou trasu až v době, kdy je potřeba, tzn. že cesta je sestavena před přenosem dat.
- **Hybridní protokoly** – kombinují vlastnosti proaktivního a reaktivního algoritmu.



Obr. 1.2: MANET směrovací protokoly

1.3 Nejvýznamnější směrovací protokoly pro Ad-hoc sítě

Nejvýznamnější směrovací protokoly pro MANET sítě jsou[1]:

- **OLSR** – je proaktivní směrovací protokol.
- **DSR (Dynamic Source Routing)** – je reaktivní směrovací protokol. Uzel, který bude posílat pakety, potřebuje znát celou cestu. K tomu používá 2 procesy:
  - proces na nalezení cesty, který funguje jako *flooding* a má dynamicky najít nové cesty,
  - proces údržby trasy, který periodicky detekuje a upozorňuje na změny v topologii.

- **AODV (Ad Hoc On-demand Distance Vector)** – je reaktivní směrovací protokol, který je kombinací protokolů DSR a DSDV (Destination Sequenced Distance Vector). Používá se:
  - pro vyhledávání trasy na požádání (on demand),
  - proces údržby trasy od DSR,
  - směrování hop-by-hop a sekvenční číslo od DSDV.
- **TORA (Temporally Ordered Routing Algorithm)** – je reaktivní směrovací protokol. Poskytuje více tras pro všechny požadované zdroj-cíl páry. Tento protokol sestaví a udržuje DAG (Directed Acycling Graph) graf. Má 3 základní funkce:
  - vytvoření trasy,
  - údržba trasy,
  - vymazání trasy.
- **GRP (Geographic Routing Protocol)** – je proaktivní směrovací protokol. Pozice všech uzlů jsou označeny GPS souřadnicemi a *flooding* je optimalizován kvadranty.

2 KVALITA SLUŽEB V MANET SÍTÍCH

Kvalita služeb (QoS) je funkce, která je schopna měřit a regulovat jednotlivé vlastnosti sítě a tím udržuje určitou kvalitu komunikace. Například v sítích, kde je QoS podporováno, se zprávy s vyšší prioritou přenáší před zprávami s nižší prioritou. Použitím QoS v ad-hoc síti nastává specifická situace, protože možnosti v oblasti základních složek QoS jsou poměrně omezené:[2]

1. **uzly se pohybují celou dobu** – díky tomu se topologie sítě celou dobu mění. Některé části sítě se dělí na části menší nebo opačně a některé části sítě se spojí. Díky této vlastnosti je nezbytná schopnost rychlé konvergence zajištěna směrovacím protokolem, aby rychle vznikaly nové cesty nebo nebyly používány ty již nefunkční.
2. **uzly mají omezené možnosti výpočetního výkonu a kapacity** – jednotlivé uzly, které tvoří MANET síť, jsou mobilní zařízení (PDA, *netbook* a *notebook*) a z toho je evidentní, že jejich zdroje nejsou používané jen na výpočet síťových událostí, ale i na splnění požadavků uživatelů.
3. **výdrž baterií** – rychlé vybití nebo optimální využívání energie v baterii velice souvisí se změnou topologie i s množstvím výpočtů síťových událostí. Jestliže je protokol z hlediska energetické náročnosti dobře nastaven, může ušetřit spoustu energie, což je pozitivní pro prodloužení doby zajištěné úrovně dané kvality služeb.
4. **nedostatek ústředního orgánu a lidského zásahu** – to je specifická vlastnost ad-hoc sítí. Z hlediska směrovacích protokolů (zajišťujících kvalitu služeb), které běží na jednotlivých uzlech, je velice důležité, aby v plynule měnící se síti udržovaly konektivitu mezi uzly a zajistily bezchybný provoz dle QoS.

2.1 Metriky

Směrovací protokol, který je používán v dané MANET síti, by měl najít právě tu cestu, která je z výše uvedených kritérií nejoptimálnější. Toto je velmi komplexní úkol a proto byly zavedeny tzv. metriky, podle kterých uzly směrují provoz anebo se rozhodují o změnách v síti. Abychom dosáhli požadovanou úroveň QoS, je potřeba počítat minimálně s následujícími síťovými parametry. Tyto parametry „metriky“ mohou být např.:[2]

- zpoždění – *(delay)*
- změna zpoždění – *(jitter)*
- šířka pásma – *(bandwidth)*
- zahozené pakety – *(dropped packets)*

Metriky kvality služeb můžeme rozdělit následujícím způsobem. Rozdělíme ji na tři třídy:[2]

- aditivní,
- konkávní,
- multiplikativní metriky.

Nechť m(A, B) je měřená metrika na lince l(A, B) mezi uzly A a B. Tedy cesta mezi A a B bude vypadat jako sekvence linek mezi uzly A, A1, A2, A3, ... An a B.[2]

Za **aditivní** metriku můžeme považovat např. zpoždění mezi A a B, protože změřené hodnoty zpoždění mezi uzly můžeme sečíst.[2]

$$m(A, B) = m(A, A1) + m(A1, A2) + m(A2, A3) + \cdots + m(An, B) \tag{2.1}$$

Za **konkávní** metriku můžeme považovat např. šířku pásma bw(A, B) mezi A a B. Z hlediska QoS je důležitá ta nejnižší hodnota (tzn. nejmenší šířka pásma) cesty, protože je třeba vědět, jestli bude nebo nebude dostatečná pro danou službu.[2]

$$m(A, B) = min\{m(A, A1), m(A1, A2), m(A2, A3), \ldots, m(An, B)\} \tag{2.2}$$

Za **multiplikativní** metriku můžeme považovat např. pravděpodobnost přijetí paketu prav(A, B) do uzlu B od uzlu A. Protože každá část cesty má individuální pravděpodobnost.[2]

$$m(A, B) = m(A, A1) \times m(A1, A2) \times m(A2, A3) \times \cdots \times m(An, B) \tag{2.3}$$

Z těchto údajů vyplývá, že šířka pásma a energie jsou konkávní metriky, cena cesty, zpoždění a změna zpoždění jsou metriky aditivní. Spolehlivost, dostupnost anebo pravděpodobnost selhání linek jsou více komplexnější problémy, proto patří mezi metriky multiplikativní.[2]

2.2 Modely kvality služeb

Modely kvality služeb jsou chápány jako architektury, které nám umožňují lepší operaci služeb v MANET síti. Nejvýznamnější modely jsou následující:[2]

- **Integrované služby** *(IntServ)* – Základní myšlenka IntServ modelu je to, že uzly musí rezervovat zdroje, aby udržovaly úroveň QoS na trase, kde provoz uskuteční.
- **Diferencované služby** *(DiffServ)* – Byly navrženy tak, aby překonaly náročnost implementace IntServ modelu.
- **Flexibilní QoS model pro MANET sítě** *(FQMM)* – První model, který byl speciálně navržen na zajištění kvality služeb v MANET sítích. Je v ní

implementována výhoda IntServ a DiffServ modelů. Funguje tak, že provoz aplikací s vyšší prioritou je zajišťován principem IntServu a provoz aplikací s nižší prioritou principem DiffServu.

- **Kompletní a účinný QoS model pro MANET sítě** *(CEQMM)* – Je velice podobný model jako FQMM, taky kombinuje vlastnosti modelů Int a DiffServ. Rozdíl je v tom, že FQMM nedefinuje použití konkrétního QoS směrovacího protokolu, ale CEQMM se definuje užíváním směrovacího protokolu QOLSR.

2.3 Směrovací protokoly pro MANET s podporou QoS

Směrovací protokoly s podporou QoS, jak je patrné z přílohy A, můžeme rozdělit do pěti skupin a následně i dalších podskupin. Těch pět skupin je:[3]

1. **Založené na topologii** – je jednou z nejpopulárnějších skupin, která rozlišuje MANET QoS protokoly. Tato skupina směrovacích protokolů zpracovává směrovací žádosti na základě toho, jak jsou distribuční cesty mezi uzly konstruovány.
2. **Objevování cesty s QoS mechanismem** – se dělí dále na tři podskupiny a jsou navrženy na základě toho, jaký mají mechanismus pro aktualizaci směrovacích informací.
3. **Založené na interakci mezi síťovou a MAC vrstvou** – jsou rozděleny dále na dvě skupiny podle toho, jak spolupracuje v daném protokolu síťová vrstva s vrstvou MAC. Existují závislé a nezávislé protokoly pro MAC vrstvu.
4. **Založené na QoS metrice** – tyto protokoly používají při rozhodování některé QoS metriky. Ve starších protokolech algoritmu se používá jenom jedna metrika, ale u novějších (hlavně těch, které byly navrženy na podporu multimediálních přenosů) se metrik používá více.
5. **Založené na základě typů zajištění QoS** – pokud jsou požadavky na QoS garantované předem na celou dobu trvání relace, protokol se řadí do podskupin tzv. tvrdé QoS. Jestli nejsou garantované na celou dobu relace, tak se protokol řadí do podskupin tzv. měkké QoS.

Většina směrovacích protokolů, které podporují QoS v MANET síti, jsou varianty jednotlivých stávajících protokolů. Hlavní vlastnosti nejvýznamnějších protokolů:

2.3.1 QOLSR

Je to proaktivní protokol OLSR s podporu QoS. QOLSR využívá směrovací zprávy HELLO a TC. Periodické vysílání HELLO zpráv slouží k prozkoumání sítě a k detekcím sousedů. TC zpráva se využívá na distribuování změny v síti a na vysílání QoS informace. Protokol zpracovává více metrik (šířka pásma, zpoždění, dále např. *loss probability*), proto se dobře využívá na přenos dat v multimediálních sítích. QOLSR umožňuje redukci vysílání směrovacích zpráv pomocí zvolení MPR uzlů. Pro činnost a na zajištění optimálních cest dle požadavků kvality služeb potřebuje distribuovat QoS informace v celé síti. V síti si uzly ještě zvolí i tzv. QMPR uzly, které mají funkci vysílat QoS informace o jednotlivých linkách, pomocí TC zpráv. V TC zprávě jsou tři pole pro informace související s kvalitou služeb:[4]

- hlavní adresa QMPR voličů
- QoS hodnoty (šířka pásma a zpoždění)
  - **dostupná šířka pásma** – 24-bitové číslo, měří se v kilobitech za sekundu. Prvních 16 bitů je nejvyšší celá část a posledních 8 nejnižších bitů představuje desetinnou část.
  - **zpoždění (Z)** – je vypočteno dle následujícího vzorce:

    $$Z = C(1 + \frac{a}{16})2^b \tag{2.4}$$

    Kde navržená konstanta C je kalibrační faktor pro výpočet zpoždění, což je 0,0625 vteřin. Proměnné „a“ a „b“ jsou nejvyšší a nejnižší čtyři bity v poli zpoždění.
- QoS hodnoty (ostatní QoS metriky)

Rozhodnutí dle šířky pásma (Bw) a zpoždění (Z) se děje následujícím způsobem. Protokol potřebuje měřit šířku pásem na všechny linky na celou cestu a vybrat nejnižší hodnotu. (A a B jsou uzly na obou koncích komunikace, Ax jsou uzly, přes které komunikace probíhá)[5]

$$Bw(A,B) = min\{Bw((A,A1), Bw((A1,A2), Bw((A2,A3), \ldots, Bw((An,B)\} \tag{2.5}$$

V případě zpoždění to je opačně. Pro měření hodnoty zpoždění se vybere nejvyšší hodnota.[5]

$$Z(A,B) = min\{Z((A,A1), Z((A1,A2), Z((A2,A3), \ldots, Z((An,B)\} \tag{2.6}$$

Pro nalezení minimální šířky pásma Δ_{Bw} a maximálního zpoždění Δ_Z si protokol začne hlídat takovou cestu (stejnou nebo s nižším počtem skoků), kde všechny hodnoty šířek pásem jsou větší nebo stejně velké jako Δ_{Bw} a kde všechny hodnoty zpoždění jsou menší nebo stejně velké jako Δ_Z.[5]

2.3.2 QAODV

Základem protokolu je reaktivní protokol AODV. Na zajištění kvality služeb bere protokol v úvaze metriky šířka pásma a zpoždění.[6]

Šířka pásma je vypočtena následujícím způsobem:[6]

$$Bw_{\text{dostupný}} = \frac{Bw \times (T_{\text{interval}} - T_{\text{rušené}})}{T_{\text{interval}}} \tag{2.7}$$

Při výpočtu šířky pásma je třeba stanovit hodnoty doba intervalu (T_{interval}) a doba rušené ($T_{\text{rušené}}$). Bw je nejvyšší možná kapacita linky, doba intervalu je čas existence spojení, doba rušené je čas, kdy je spojení jakýmkoli způsobem obsazeno.[6]

Zpoždění (Z) je vypočteno následujícím způsobem:[6]

$$\mathrm{Z}_i = \frac{l_i + Z_0^{i-1} \times (\eta_i - \mu_i)}{\eta_i}, i = 1, \ldots m \tag{2.8}$$

Výpočet pro uzel „i“ probíhá tak, že do vzorce 2.8 se doplní hodnoty l_i délka fronty, η_i rychlost vysílání paketů, μ_i rychlost přijímání paketů a Z_0 zpoždění na předcházející lince na cestě.[6]

Protokol QAODV redukuje zpoždění a zvýší poměr doručení paketů při vysokém zatížení a středně vysoké mobilitě. Z těchto důvodů produkuje QAODV větší směrovací provoz oproti tradičnímu AODV.[6]

2.3.3 CEDAR

Protokol je navržený pro splnění QoS směrování pro malé a střední ad-hoc sítě (desítky až stovky uzlů). Dynamicky sestaví jádro sítě a potom začne propagovat stabilní linky k uzlům, které jsou v jádru. Výpočet cesty začíná na požádání a vykonávají jej uzly tvořící jádro sítě. Protokol má tři hlavní komponenty:[3]

1. **volba jádra** – volba uzlů, které budou tvořit jádro sítě.
2. **propagace stavu linek** – QoS směrování je zajištěno s propagací šířky pásma stabilních linek, které spojují uzly s jádrem sítě. Hlavní myšlenka je v tom, že linky s vysokým výkonem se musí distribuovat v celé síti, ale informace o slabé lince musí zůstat v lokální části sítě.
3. **výpočet cesty** – probíhá v jádru sítě. Provádí se na požádání („*on demand*“), kdy najdou trasu ještě před přenosem dat.

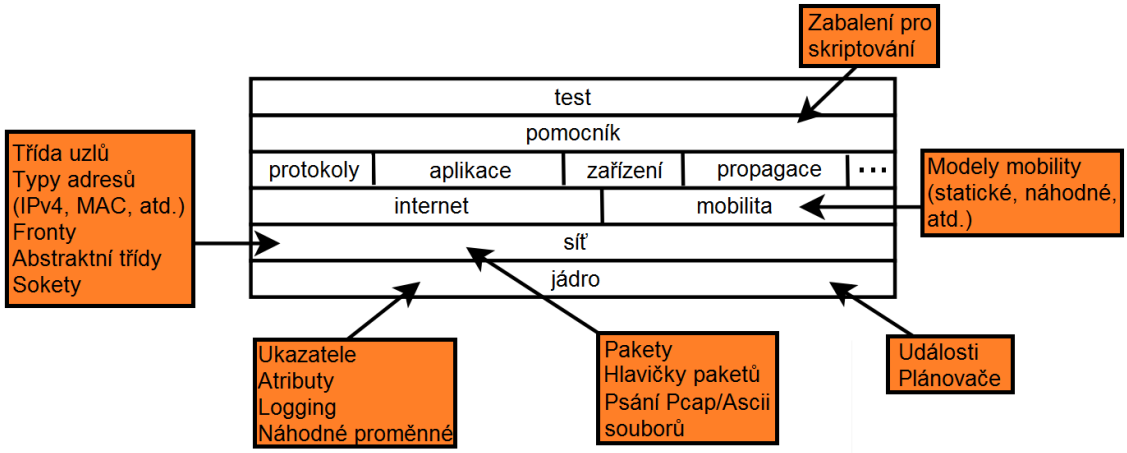
Směrovací protokol CEDAR používá metriku šířka pásma pro zajištění kvality služeb. Tady výpočet šířky pásma neprobíhá, protože na všech linkách MAC vrstva aktuální dostupnou šířku pásma odhaduje. Jestli se na jedné lince dostupná šířka

pásma zvýší, protokol ji označuje jako „linka s vysokým výkonem“, a je propagována v celé síti. V opačném případě pokud se dostupná šířka pásma na jedné lince sníží pod určitou úroveň, je velice rychle propagováno, aby bylo co nejrychleji zachráněno spouštění provozu na lince, která již nemá takovou kapacitu, než měla.[7]

3 SIMULAČNÍ PROSTŘEDÍ NETWORK SIMULATOR NS-3

Simulační prostředí *ns-3* se skládá ze dvou částí, jádro a modely simulací. Obě části jsou implementovány v programovacím jazyku C++. Simulační program je navržen jako knihovna, která je staticky nebo dynamicky připojena do hlavního C++ programu, který definuje topologii sítě a spouští simulátor.[8]

Zdrojový kód simulačního programu *ns-3* je organizován v adresáři `src` a je popsán na následujícím obrázku.[8]



Obr. 3.1: Struktura programu[8]

3.1 Simulační modely

Základem celého projektu *ns-3* bylo to, aby se jádro simulačního programu rozvíjelo směrem ke snazší použitelnosti, laditelnosti a aby bylo vše dostatečně popsáno a zdokumentováno. Vlastnosti simulátoru dovolují práci na všech typech reálných sítích, ale většina jeho uživatelů se zaměřuje na simulaci sítí bezdrátových, které zahrnují modely pro Wi-Fi, WiMAX a různé statické a dynamické směrovací protokoly jako OLSR a AODV pro IP aplikace.[8]

3.2 Jádro simulátoru

Jádro je v adresáři `src/core` a zahrnuje ty součásti (atributy), které jsou společné pro všechny protokoly, hardware a environmentální modely. Pakety jsou základní objekty simulačního prostředí. Proto tvoří pakety další část jádra, které jsou uloženy

v adresáři `src/network`. Tyto dvě části vytváří univerzální jádro simulačního programu, které můžeme použít nejen v sítích Internetových, ale ve všech ostatních typech. (Podporuje výzkum nejen v sítích IP ale i v non-IP sítích.)[8]

3.3 Dokumentace - Doxygen

O dokumentaci simulačního programu *ns-3* se stará tzv. *Doxygen*. *Doxygen*, který se obvykle používá pro dokumentaci API a organizuje dokumentaci do určitých částí, které můžeme rozdělit do čtyř skupin:[9]

- Moduly
- *Namespaces*
- Třídy
- Soubory

V těchto čtyřech částech jsou popsány vlastnosti veškerých příkazů, tříd a jiných užitečných věcí. Další informací o Doxygenu můžeme najít na této webové stránce: http://www.nsnam.org/docs/release/3.15/doxygen/index.html.[9]

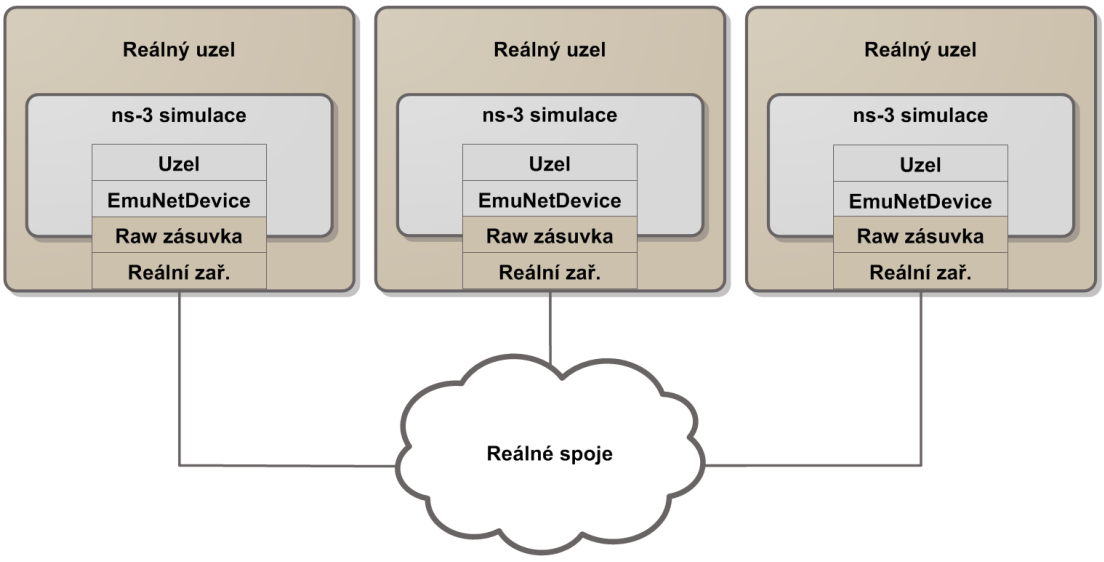
Další materiály a dokumentace můžeme najít ve formě pdf dokumentů na stránce `http://www.nsnam.org`. Tyto dokumenty se zaobírají tutoriály simulátoru, popsáním jádra a jiných částí programu.[9]

3.4 Metody zapojení reálných zařízení do simulace

Simulátor *ns-3* byl navržen pro integraci do různých reálných testovacích prostředí a do prostředí virtuálních strojů. Existují dvě metody pro realizaci možností zapojení reálných zařízení do simulace. První z možností se jmenuje *EmuNetDevice*, což umožní posílat data mezi reálnými zařízeními, na kterých běží simulace. Druhá varianta se jmenuje *TapNetDevice*, která byla navržena pro zapojení reálných zařízení do *ns-3* simulace takovým způsobem, aby z hlediska simulačního programu byli jediní ze simulovaných uzlů. Simulace mohou být konstruovány s libovolnou kombinací *Emu* nebo *Tap* zařízeních.[9]

3.4.1 Metoda *EmuNetDevice*

Základem této metody je, že na každém zařízení běží simulace. V tomto smyslu každá simulace je podmnožinou globální simulace. Tyto reálné uzly na místě *ns-3* kanálu jsou propojeny pomocí takového reálného spoje, který se používá v běžné simulaci. To umožňuje *ns-3* aplikacím a protokolovým zásobníkům být připojenými k reálným uzlům a komunikovat přes skutečná média.[9]



Obr. 3.2: Ukázka struktury metody *EmuNetDevice*

Metoda *EmuNetDevice* umožňuje simulačnímu uzlu posílat a přijímat pakety přes reálné sítě. Emulované zařízení se spoléhá na reálné rozhraní, které je zapnuté a je přepnuto do promiskuitního režimu. Tím se otevře *Raw* zásobník[1] a váže se na toto rozhraní.[9]

Je implementován *MAC spoofing* pro oddělení provozu simulační sítě od jiného provozu sítě, který muže proudit od a do počítače. OUI[2], část MAC adresy, bude generován (ve výchozím nastavení) s hodnotou 00:00:00. Tento kód dodavatele není přiřazen žádné organizaci a proto by neměl být v rozporu se žádným jiným skutečným zařízením. Pro větší simulace je možnost použít alokaci pro MAC adresy.[9]

Hlavním cílem této metody je, aby generovala opakovatelné experimentální výsledky v reálném síťovém prostředí, které zahrnuje všechny funkce simulátoru *ns-3* jako např.: trasování, logování, vizualizace a nástroje pro statistiky.[9]

3.4.2 Metoda *TapNetDevice*

Druhá metoda, jak implementovat reálné zařízení do *ns-3* simulace, se jmenuje *TapNetDevice*. Tato metoda má inverzní myšlenku oproti první možnosti implementace *EmuNetDevice*.[9]

Tato metoda byla navržena pro testování a analýzu reálných aplikací a protokolů, zvláště pro hlídání jejich chování v přítomnosti velkých simulovaných `ns-3` sítí. Je nejlépe použitelná ve virtuálním prostředí, kde je jeden reálný host a v něm více běžících strojů virtuálních. Jak je vidět z obrázku3.3, simulace `ns-3` běží na středním virtuální stroji. Tato simulace má více uzlů, aplikací a běžících protokolů, které komunikují mezi sebou simulovanou linkou tzv. kanál.[9]

Nahoře vpravo a vlevo jsou virtuální stroje, na kterých běží operační systém Linux, potřebné aplikace a protokoly pro komunikaci. Tyto virtuální stroje jsou zapojené do simulace pomocí *Tap rozhraní* a v simulaci jsou reprezentovány pomocí tzv. proxy uzlu. Takovým způsobem je simulováno to, aby virtuální stroje byly připojeny k `ns-3` kanálu.[9]

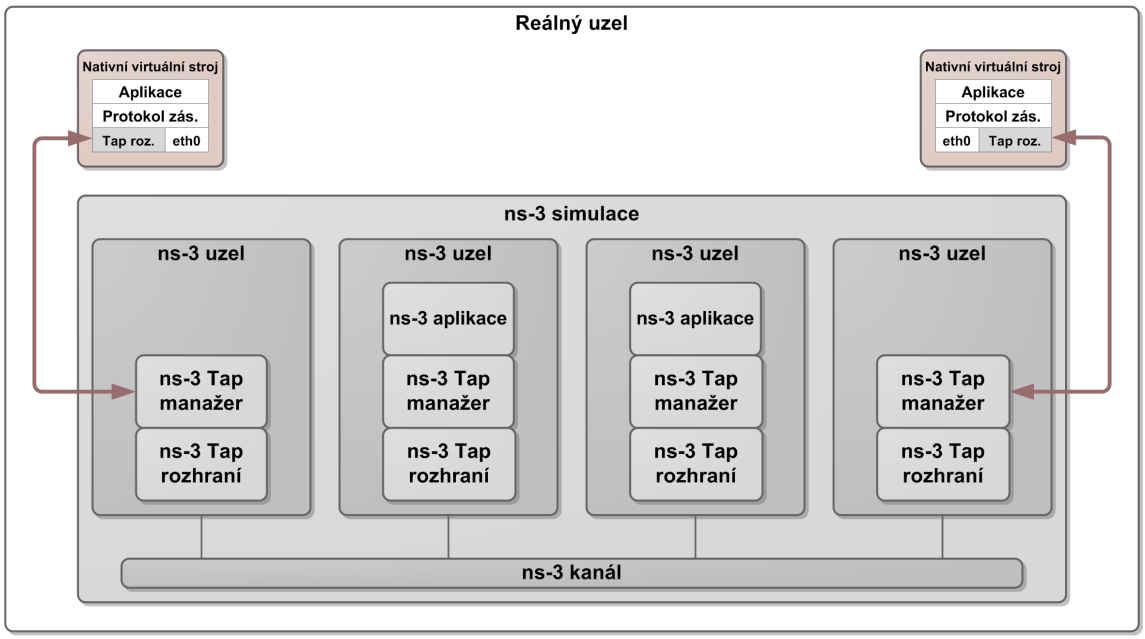
Jestli jsou dvě *Tap rozhraní* propojené, nazývá se to *Tap most* (*Tap bridge*). Existují tři základní provozní režimy tohoto přístroje, které jsou k dispozici uživatelům. Tyto tři režimy jsou následující:

- *ConfigureLocal* – Konfigurace pomocí simulátoru
- *UseLocal* – Použití existující konfigurace
- *UseBridge* – Použití existující konfigurace pomocí mostu

[1] *Raw* zásobník (*Raw socket*) umožňuje přímé odesílání a přijímání paketů.

[2] OUI (*Organizationally Unique Identifier*) – Unikátní 24 bitové číslo (identifikátor), které jednoznačně identifikuje dodavatele, výrobce nebo organizace.

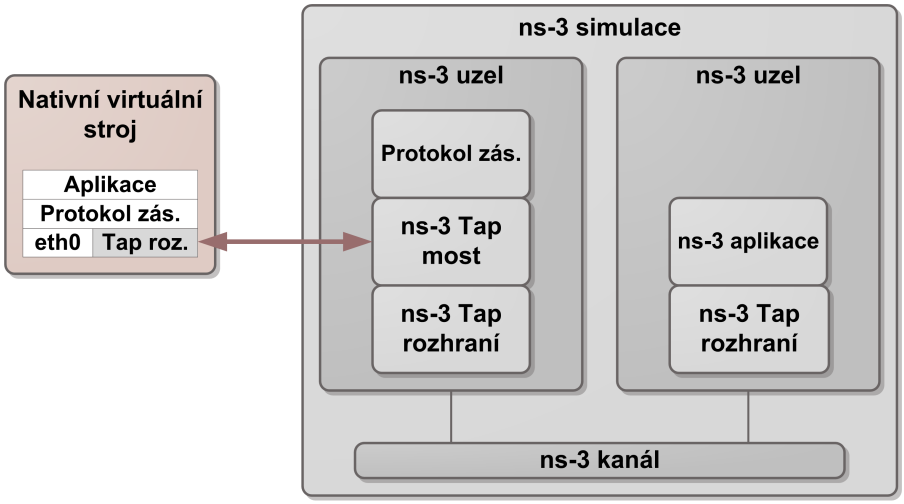
První část názvu těchto režimů identifikuje, kdo byl měl být zodpovědný pro vytvoření a konfiguraci rozhraní.[9]



Obr. 3.3: Ukázka struktury metody *TapNetDevice*

Režim *ConfigureLocal*

V tomto režimu konfigurace *Tap* rozhraní je `ns-3` centrické a je výchozím provozním režimem *Tap* mostu.[9]



Obr. 3.4: Ilustrace režimu *ConfigureLocal*

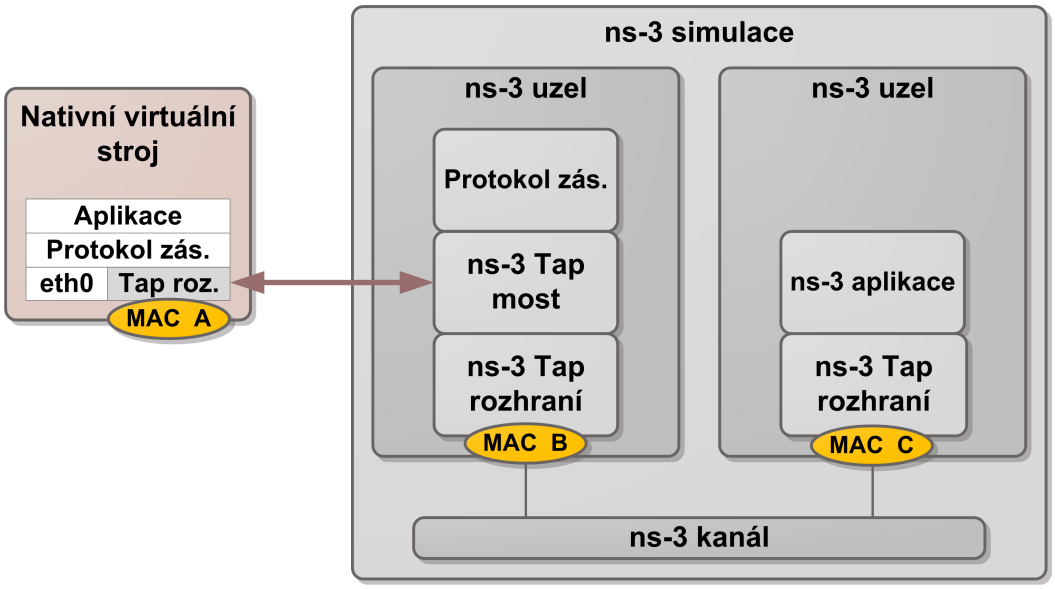
Informace o konfiguraci je převzatá ze zařízení z `ns-3` simulace a tak je zajištěno, aby *Tap* rozhraní bylo vyhovující pro simulátor. Takovým způsobem simulátor dosáhne, aby virtuální stroj byl součástí simulace jako uzel. To je znázorněno na obrázku 3.4.[9]

Tap rozhraní se v simulovaném uzlu objeví jako by bylo nahrazeno rozhraním virtuálního počítače. To se projeví tak, že simulace vytvoří *Tap* rozhraní na virtuálním stroji a nastaví IP a MAC adresy rozhraní tak, aby odpovídaly hodnoty k přiřazenému simulovanému *Tap* rozhraní.[9]

Tato konfigurace se projeví bez dohledu uživatele. Díky tomu, *Tap* rozhraní bude vytvořeno a nakonfigurováno (IP a MAC adresy) pomocí výchozích hodnot.[9]

Režim *UseLocal*

Tento režim je poměrně podobný předchozímu režimu. Významný rozdíl je ten, že jak již z názvu vyplývá, *Tap* most použije existující konfiguraci *Tap* rozhraní. Tento režim je zvláště užitečný, pokud virtualizační systém automaticky vytvoří *Tap* zařízení a *ns-3* se používá jen k poskytování simulovací sítě pro tyto zařízení.[9]

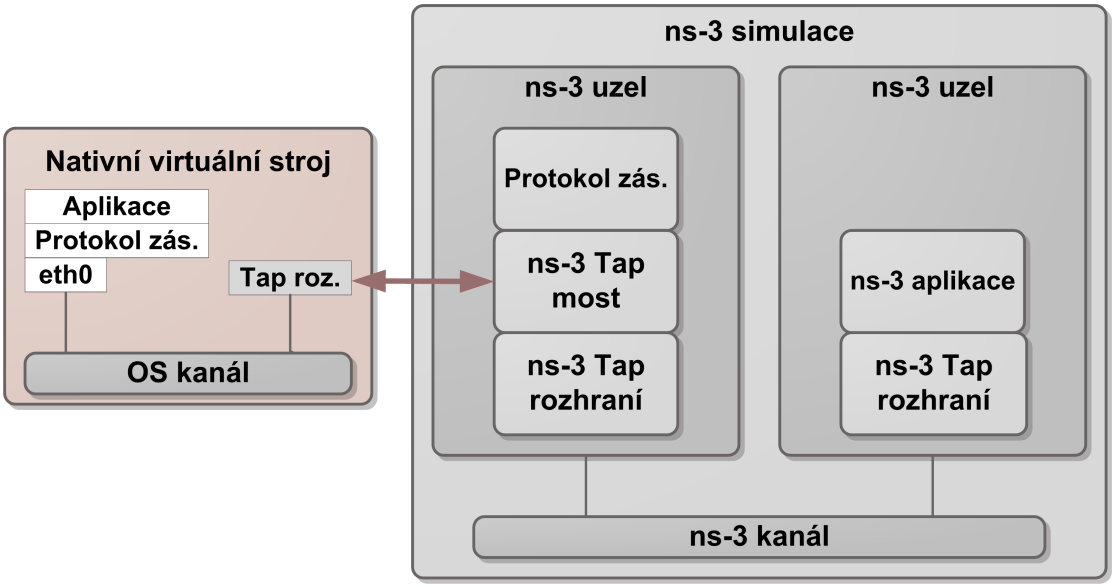


Obr. 3.5: Ilustrace režimu *UseLocal*

V tomto případe předkonfigurovaná MAC adresa *Tap* rozhraní (MAC A) na virtualizovaném systému nemusí být stejná jako u můstkového rozhraní (MAC B), jak je uvedeno na obrázku 3.4 výše.[9]

Režim *UseBridge*

Tento režim je více náročnější z hlediska síťového nastavení na Linuxu. Opět platí, že předpona „Použít“ naznačuje, že *Tap* most použije existující konfiguraci, ale v tomto případě bude most logicky rozšiřovat Linux most do simulace `ns-3`. To je znázorněno na obrázku 3.6 níže.[9]



Obr. 3.6: Ilustrace režimu *UseBridge*

V tomto případe počítač s linuxovými aplikacemi, protokoly, atd. je připojený k `ns-3` simulované síti takovým způsobem, aby se s *Tap* rozhraním komunikovalo přes Linux most a to celé se jeví jako skutečná síť mezi Linuxem a *Tap* rozhraním.[9]

4 DOSTUPNÉ IMPLEMENTACE SMĚROVACÍHO PROTOKOLU AODV

Pro zpracování dané problematiky existuje více řešení. Možnosti implementací směrovacího protokolu AODV do reálných zařízení se liší v tom, do jakého operačního systému se implementace provádí. Všechny možnosti byly rozvíjeny nezávisle na sobě a všechny řeší jeden jediný úkol – implementaci protokolu AODV.

4.1 Implementace do JAVA – JAdhoc

Projekt popisuje implementaci směrovacího protokolu AODV do operačních systémů Windows a Linux pomocí Javy.[10]

Projekt JAdhoc byl vyvíjen v jazyce JAVA pro obě platformy a využívá Jpcap na zachytávání paketů. Pro realizaci unicastového a všesměrového vysílání se využívají knihovny tříd JAVA. Časovače, které AODV používají, jsou také řešeny pomocí třídy z knihovny. Pro detekce výpadků spoje se používají Hello zprávy.[10]

4.2 Windows

V této podkapitole budou zpracovány současné možnosti implementace směrovacího protokolu AODV do operačního systému Microsoft Windows.

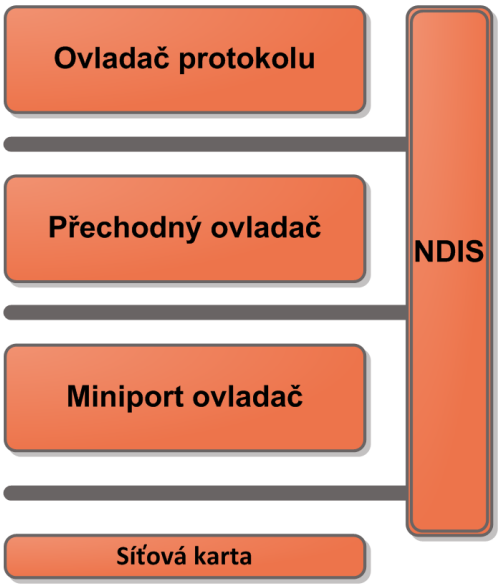
4.2.1 Implementace na uživatelské úroveni (*User Level Implementation*)

Implementace na uživatelské úrovni je vyvinuta pro testování algoritmu AODV na skutečných mobilních zařízeních se skutečným operačním systémem a dává nám reálnou představu o užití a výkonu algoritmu implementovaného na úrovni kernelu. To nám dává možnost provádět simulace algoritmu se zkušebním nastavením. Takové simulace poskytují realističtější výsledky než simulace v *Network Simulatoru*. Tato implementace ukazuje, jak byl integrován protokol do Windows CE a zároveň aby protokol bylo možné spustit přímo na datové vrstvě a aplikace mohly být postavené na tomto protokolu.[11]

4.2.2 Přechodný ovladač NDIS (*Network Driver Interface Specification*)

Tato metoda byla vyvíjena tak, aby nebylo potřeba změnit ovladač TCP/IP protokolu a byla zajištěna snadná instalace a distribuce ovladače. [11] Jako například WinAODV.

Celá metoda je postavená na NDIS rozhraní s více ovladači. Jak je vidět na obr. 4.1, rozhraní NDIS je umístěno na vyšší úrovni této architektury a slouží na komunikaci s TCP/IP protokolem. Ve střední úrovni se nachází přechodný a miniport ovladač. Tyto dva ovladače zajišťují komunikaci mezi vyššími a nižšími úrovněmi. Pro vyšší vrstvy se přechodný ovladač chová jako miniport ovladač a pro nízké vrstvy zase opačně.[11]



Obr. 4.1: NDIS architektura

Na obrázku můžeme vidět, že z hlediska komunikace je přechodný ovladač na klíčovém místě. Uvnitř v ovladači běží operace rozebírání, které slouží např. na filtrování nebo na šifrování a dešifrování paketů.[11]

4.3 Linux

Tato podkapitola se zabývá implementací AODV protokolu do platformy UNIX/Linux. Existují dva typy implementace:[12]

4.3.1 Démoni v uživatelském prostředí (*User Space Daemons*)

Základem tohoto typu implementace je proces tzv. démon, což je počítačový program, který běží na pozadí a není pod přímou kontrolou uživatele.

Mad-Hoc

Byl to první typ implementace protokolu AODV, který běžel na Linuxu s kernelem 2.2. Nevýhodou bylo, že nepodporoval multicast. Využíval metodu pro zachycení paketů libpcap. Po první verzi se dále nerozvíjel s protokolem AODV, proto dnes již není podporován.[12]

AODV-UU - *Uppsala University*

AODV-UU je možnost implementace, která byla vyvinuta na univerzitě Uppsala. Běží na Linuxu kernel 2.4 nebo 2.6. Pro podporu multicastu byl vytvořen opravný balíček. Protokol AODV je implementován pomocí démonu a dvou kernel modulů (kaodv a ip_queue_aodv). Využívá Netfilter knihovnu pro zachycení paketů.[12]

UCSB AODV – University California, Santa Barbara

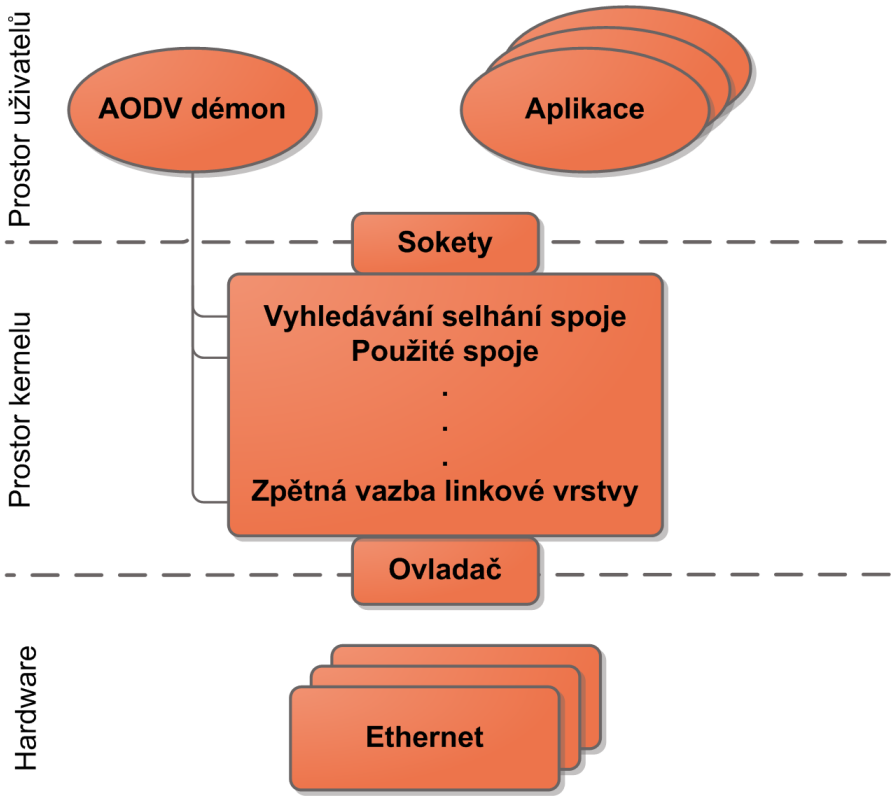
UCSB je úplně stejná metoda implementace směrovacího protokolu AODV do Linuxu jako předchozí AODV-UU.[12]

UIUC AODV – *University of Illinois, Urbana-Champaign*

Implementace typu UIUC využívá podporující knihovny ASL (*Ad-hoc Support Library*). ASL knihovna byla navržena pro poskytnutí všech služeb, které ad-hoc směrovací protokoly potřebují. Hlavním cílem této knihovny je odstranění nevýhod složitosti předchozích implementací a snadné vyvíjení jiných ad-hoc směrovacích protokolů.[12]

4.3.2 Kernel modul

Implementace se realizuje pomocí vytvoření nového kernel modulu.[12]



Obr. 4.2: Architektura modifikace kernelu

Implementace NIST

Je jediným řešením tohoto typu implementací a byla optimalizována pro Linux kernel typu 2.4. Pro úplnou realizaci implementace směrovacího protokolu AODV byl napsán kernel modul. Podporuje multicast a Internet zprostředkovává na více skoků. Využívá Netfilter pro zachycení paketů, což je část kernelu 2.4. Informuje uživatele o aktuálních spojích a vkládá do souboru statistiky.[12] Vylepšení této metody implementace AODV protokolu je projekt MAODV.[13]

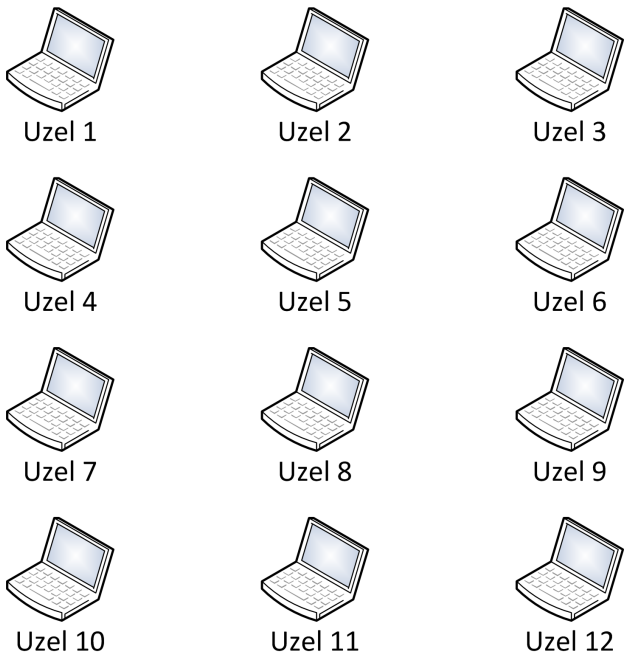
5 SIMULAČNÍ SKRIPT SE ZAMĚŘENÍM NA QOS

Tato kapitola se bude zabývat implementací směrovacího protokolu AODV v simulačním prostředí *Network Simulator* ***ns-3*** se zapnutou podporou kvality služeb *QoS*. Celá implementace byla vykonána pod operačním systémem Ubuntu 12.04. Verze simulátoru byla `ns-3.14`. Simulační program byl vytvořen v softwarovém vývojovém prostředí *Eclipse*.

5.1 Základní vlastnosti programu

Po otevření vývojového prostředí byl vytvořen projekt následujícím způsobem: `File` ↦`New`↦`Project`↦`C/C++`↦`C++ Project` s názvem `ns-3-dev`.

V programu `ns-3` byla vytvořena MANET síť se zavedenou podporou QoS. V topologii je dvanáct uzlů v mřížkovém rozdělení, jako na obr. 5.1. Každý uzel se pohybuje náhodně a směruje dle směrovacího protokolu AODV. Na kontrolu konektivity byla naprogramována funkce, která zajistí ping na 3. uzel od uzlů 1, 6, 7 a 8. Byly vytvořeny dvě aplikace FTP a VoIP. Pro oba provozy je zdrojový uzel *Uzel 3* a koncový je *Uzel 7*. Relace FTP a VoIP trvá 20 vteřin. Celková doba simulace je 30 vteřin.



Obr. 5.1: Topologie MANET sítě

Simulace je sestavena dle následujících metod:

- `CaseRun()` – řídí celou simulaci. Spouští metody pro sestavení topologie a vykonávání aplikací. Řeší generování výsledků simulace.
- `Init(int argc, char **argv)` – aktivuje logování do konzole.
- `VytvoreniUzlu()` – vytvoří dvanáct uzlů, fyzickou vrstvu, kanál a zavede podporu QoS.
- `VytvoreniZarizeni()` – vytvoří zařízení, instaluje směrovací protokol AODV, přiřadí IP adresy a definuje vlastnosti pohybu uzlů.
- `VytvoreniPing()` – vytvoří aplikaci *Ping* pro dané uzly.
- `Aplikace_Ftp()` – vytvoří a definuje vlastnosti aplikací FTP.
- `Aplikace_Udp()` – vytvoří a definuje vlastnosti aplikací VoIP.

5.2 Popis částí programu

5.2.1 Inicializace

Na začátku programu byly volány z knihovny `ns-3` hlavičkové soubory, které během simulace používají jednotlivé části programu. Z těch nejdůležitějších jsou:

```
13 #include "ns3/wifi-module.h"
```

V tomto hlavičkovém souboru jsou vlastnosti týkající se bezdrátových sítí a uzlů. Každý hlavičkový modul je připojen pomocí klasického preprocesoru `#include"..."`.

```
14 #include "ns3/mobility-module.h"
```

Modul pohybu slouží pro načtení jednotlivých typů mobility. Dva z hlavních typů jsou konstantní (kde se uzly nepohybují, např. přístupový bod) a náhodná mobilita (kde se uzly pohybují náhodným směrem).

```
18 #include "ns3/netanim-module.h"
```

Slouží na volání příslušenství programu *NetAnim*. Po vyexportování jednotlivých parametrů (např. pozice uzlů na souřadnicový systém X a Y) do souboru typu `.xml`, můžeme pomocí tohoto programu sledovat změny parametrů v čase. V tomto případě jak se uzly pohybují.

```
17 #include "ns3/aodv-helper.h"
```

Volba a implementace směrovacího protokolu AODV začne tímto krokem. Tento modul nám pomůže načítat jednotlivé vlastnosti protokolu jako např. typy a parametry paketů.

Simulační program v `ns-3` začne vždy voláním jmenného prostoru. Hned na začátku ještě spouštíme logování. To se uskuteční použitím příkazů:

```
34 using namespace ns3;
35 NS_LOG_COMPONENT_DEFINE ("potfay_aodv_log");
```

5.2.2 `CaseRun()`

Tato metoda slouží na spouštení všech metod, které jsou tvořeny pro to, aby vytvořily uzly, topologii a aplikaci. To probíhá následujícím způsobem:

```
124 VytvoreniUzlu();
125 VytvoreniZarizeni();
126 VytvoreniPing();
127 Aplikace_Ftp();
128 Aplikace_Udp();
```

NetAnim

Jak již bylo zmíněno, tato metoda slouží i pro vygenerování výsledků. Další řádky slouží na to, aby program vygeneroval do souboru aktuální hodnoty pozice uzlů. Vytvoří soubor ve formátu XML, nastaví popis uzlů na *Uzel* a povolí uložení informace o paketech.

```
138 std::string animFile = "aodvPotfay_NetAnim.xml";
139 AnimationInterface::SetNodeDescription(uzly, "Uzel");
140 AnimationInterface anim(animFile);
141 anim.EnablePacketMetadata(true);
```

Flow Monitor

Metoda, která analyzuje vlastnosti sítě během simulace se nazývá *Flow Monitor*. Tato funkce sbírá hodnoty zpoždění, jitteru a ještě další vlastnosti, které se objevují mezi koncovými stanicemi. Další řádky definují existenci *Flow Monitor*-u v programu:

```
152 flowMon = flowMonHelper.InstallAll();
153 Ptr<Ipv4FlowClassifier> ipv4clas = DynamicCast<Ipv4FlowClassifier>(
    flowMonHelper.GetClassifier());
```

Výsledky a hodnoty jsou vyexportovány do souboru typu XML a jsou hlídány pakety, které se zdají být ztraceny.

```
167 flowMon->CheckForLostPackets();
168 flowMon->SerializeToXmlFile("aodvPotfay_FlowMon.flowmon", true,
    true);
```

Následující řádky povolí vypisování daných výsledků *Flow Monitor*-u do konzoly:

```
170 std::map<FlowId, FlowMonitor::FlowStats> stats = flowMon->
    GetFlowStats();
171   for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i =
        stats.begin(); i != stats.end(); ++i) {
172     Ipv4FlowClassifier::FiveTuple t = ipv4clas->FindFlow(i->first);
173     std::cout << "Flow " << i->first - 2 << " (" << t.sourceAddress
          << " -> " << t.destinationAddress << ")\n";
174     std::cout << "Tx Bytes:   " << i->second.txBytes << "\n";
175     std::cout << "  Rx Bytes:   " << i->second.rxBytes << "\n";
176     std::cout << "  Throughput: " << i->second.rxBytes * 8.0 / 8.0
        / 1024 / 1024 << " Mbps\n";
177     std::cout << "  Tx Packets:   " << i->second.txPackets << "\n";
178     std::cout << "  Rx Packets:   " << i->second.rxPackets << "\n";
179     std::cout << "  Delay Sum:   " << i->second.delaySum << "\n";
180     std::cout << "  Average Delay:   " << i->second.delaySum / i->
        second.rxPackets << "\n";
```

Řízení simulace

```
134 Simulator::Stop(Seconds(simTime));
135 Simulator::Run();
136 Simulator::Destroy();
```

Následující příkazy slouží pro vytvoření souboru typu `.xml`. Do prvního souboru byla ukládána informace pro funkci programu *NetAnim* a do druhého pro funkci *Flow Monitor*-u, podle níž jsou sestaveny statistiky a grafy.

```
59 std::string netAnimSoubor = "manetrouting.xml";
60 std::string flowMonSoubor = "manetroutingFlow.xml";
```

5.2.3 Init(int argc, char **argv)

Funkce `Init` slouží na zprovoznění a povolení vypsání logů do konzole.

```
108   if (verbose) {
109     LogComponentEnable("potfay_aodv_log", LOG_LEVEL_ALL);
110   }
```

5.2.4 VytvoreniUzlu()

Vytvoření nových uzlů se uskuteční pomocí třídy `NodeContainer`. Tato třída se hlavně používá v takových případech, když je potřeba vytvořit velké množství stejných uzlů v síti. [9]

```
65 NodeContainer uzly;
```

```
188 uzly.Create(12);
```

Následující příkaz slouží na deklaraci kanálu. Třída `YansWifiChannelHelper` má za úkol vytvořit a spravovat bezdrátové kanály dle „YANS (Yet Another Network Simulator)“ modelu.[9]

```
68 YansWifiChannelHelper kanal;
```

```
190 kanal = YansWifiChannelHelper::Default();
```

Fyzická vrstva je definována stejným způsobem jako kanál, kromě toho, že je použita jiná třída. Potom je ještě logicky spojena fyzická vrstva s novým vytvořeným přenosovým kanálem.

```
69 YansWifiPhyHelper fyz;
```

```
191 fyz = YansWifiPhyHelper::Default ();
192 fyz.SetChannel(kanal.Create());
```

Třída `WifiHelper` pomůže vytvořit a konfigurovat vlastnosti bezdrátových zařízení. Na fyzické vrstvě byl nastaven standard 802.11b a algoritmus na řízení rychlosti.[9]

```
70 WifiHelper wifi;
```

```
194 wifi.SetStandard(WIFI_PHY_STANDARD_80211b);
195 wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "
    DataMode", StringValue("DsssRate1Mbps"));
```

Zapnutí a vypnutí podpory QoS

Následujícím příkazem byla aktivována kvalita služeb v simulaci. Třída `QosWifiMac Helper` může vytvářet více druhů řízení přístupu k médiu.[9] Jeden z těch typů je `AdhocWifiMac`, který se používá i tady.

```
71 QosWifiMacHelper qos;
```

```
198 qos = QosWifiMacHelper::Default();
199 qos.SetType("ns3::AdhocWifiMac");
```

Na úplné vypnutí kvality služeb slouží třída `NqosWifiMacHelper`.[9] Ta se liší od třídy sloužící na zapnutí jenom v prvním znaku (velké N), což můžeme pochopit jako negaci. Ukázka na zprovoznění:

```
72 NqosWifiMacHelper qos;
```

```
199 qos = NqosWifiMacHelper::Default();
```

Editování hlavičkových souborů

Editace hlavičkových souborů byla třeba pro zprovoznění třídění paketů dle QoS. Tyto změny byly provedeny v následujících souborech.

První takový soubor byl `onoff-application.cc`. První změna byla připojení hlavičkového souboru, který definuje priority, pro roztřídění provozů.

```
41 #include "ns3/qos-tag.h"
```

Další řádky vytvářejí nový nastavitelný atribut pro všechny On-Off aplikace.

```
89 .AddAttribute ("AccessClass","The access class: AC_BE, AC_VO, AC_VI
   , AC_BK",
90                   UintegerValue (0),
91                   MakeUintegerAccessor (&OnOffApplication::qosTag)
                        ,
92                   MakeUintegerChecker<uint8_t> ())
```

Přidávání proměnné `qosTag`:

```
109 qosTag = 0;
```

Poslední změna v tomto souboru řeší doplnění informace o třídě do paketu:

```
256 packet->AddPacketTag (QosTag (qosTag));
```

Druhý takový soubor je `onoff-application.h`, v něm byly vytvořeny pouze nové proměnné:

```
148 uint8_t         qosTag;
```

5.2.5 VytvoreniZarizeni()

Jak již bylo řečeno u `NodeContainer`-u, kontejnery se používají pro vytvoření velkého množství stejných objektů.[9] V tomto případě jsou vytvořena stejná zařízení. Další příkaz slouží na instalaci zařízení do uzlů s vytvořenou fyzickou vrstvou a podporou QoS.

```
66 NetDeviceContainer zarizeni;
```

```
206 zarizeni = wifi.Install (fyz, qos, uzly);
```

AODV

Třída `AodvHelper` pomůže přiřadit k uzlům směrovací protokol AODV.

```
74 AodvHelper aodv;
```

Pomocí následujících tříd se řeší funkcionalita protokolů IP, ICMP, TCP a UDP. Spojí jejich vlastnosti a přiřadí k uzlům. Před přiřazením jsme spojili s protokoly i protokol AODV.

```
73 InternetStackHelper stack;
```

```
208 stack.SetRoutingHelper(aodv);
209 stack.Install(uzly);
```

Přiřazení IP adres

Třída `Ipv4AddressHelper` je nejjednodušším způsobem, jak přiřadit IP adresy uzlům a rozhraním. Pokud všechny uzly nejsou ve stejné síti a potřebujeme více adresních prostorů, tak je třeba použít třídu `Ipv4AddressGenerator` (to řeší komplikovanější problémy).[9] Rozhraní byly vytvořeny následujícím příkazem:

```
67 Ipv4InterfaceContainer rozhrani;
```

Poslední řádek řeší hlavní myšlenku MANET sítí. Sestaví směrovací databázi a inicializuje směrovací tabulky jednotlivých uzlů. Udělá z každého uzlu směrovače.

```
211 Ipv4AddressHelper adresy;
212 adresy.SetBase("10.0.0.0", "255.255.255.0");
213 adresy.Assign(zarizeni);
214 rozhrani = adresy.Assign(zarizeni);
215 Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

Nastavení pohybu uzlů

Pohyb uzlů se řeší následujícím způsobem. Pomocí třídy `MobilityHelper` je vytvořena proměnná `mobilita`, která slouží na spojení vlastností týkajících se pohybu uzlů.

```
75 MobilityHelper mobilita;
```

Jsou definované počáteční pozice a poté je nastaven model nebo typ pohybu. Typ pohybu byl zvolen náhodný, protože ten popíše nejlépe chování uzlů v sítí typu MANET.

```
223 mobilita.SetMobilityModel("ns3::RandomWalk2dMobilityModel", "Bounds
    ", RectangleValue(Rectangle(-50, 50, -50, 50)));
224 mobilita.Install(uzly);
```

Vygenerování pcap souborů

Soubory `pcap` slouží k uložení paketů vysílaných během simulace. Tyto soubory používá program např. *WireShark*, v němž byla i v této práci vykonána analýza paketů. Následující příkaz řeší povolení exportování do `pcap` souborů na všechny uzly:

```
304 fyz.EnablePcapAll(std::string("aodvPotfay"));
```

5.2.6 `VytvoreniPing()`

Byla vytvořena aplikace ping pro kontrolu konektivity. Třída `V4PingHelper` vytvoří jeden nebo více aplikací ping a přiřazuje je uzlům. Následující příkaz slouží na označení uzlu, se kterým bude konektivita kontrolována.

```
310 V4PingHelper ping(rozhrani.GetAddress(3));
```

Následující řádek řeší vytvoření aplikace na *Uzel 7*.

```
313 ApplicationContainer appPing0 = ping.Install(uzly.Get(7));
```

Aplikace začne hned na začátku simulace a skončí v čase 10 sekund.

```
317 appPing0.Start(Seconds(0.01));
318 appPing0.Stop(Seconds(10));
```

5.2.7 Aplikace

Pro testování celé sítě a kvality služeb byly vytvořeny aplikace, které generují provoz. Zdrojový uzel těchto provozů byl zvolen `Uzel 3` a cílový `Uzel 7`. První aplikace byla zvolena pro to, aby byla síť zatížena a druhá pro to, aby na ní byl měřen vliv nastavení a vypnutí podpory QoS.

FTP provoz proběhne na `port(21)` a VoIP na `port2(5060)`. Tato nastavení byla globálně nastavena následujícím způsobem:

```
93 PotfayAodv::PotfayAodv() :
94         verbose(true), simTime(30), pcap(true), bytesTotal(0),
               packetsReceived(0), port(21), port2(5060) {
95 }
```

Tyto aplikace v programu `ns-3` se definují pomocí On-Off aplikací.

`Aplikace_Ftp()`

První z těchto aplikací je FTP provoz. Její QoS třída byla nastavena na co nejmenší tzn. *Best Effort* z takového důvodu, aby po zapnutí podpory QoS, byl potlačen.

Následující řádky vytvářejí proměnné pro uzly zdroj a cíl provozu. Je i vyčítána IP adresa cílového uzlu.

```
253 Ptr<Node> ftpSource = uzly.Get(3);
254 Ptr<Node> ftpSink = uzly.Get(7);
255 Ipv4Address remoteAddr = ftpSink->GetObject<Ipv4>()->GetAddress(1,
        0).GetLocal();
```

Další řádky vytvářejí aplikaci On-Off. K aplikaci jsou přiřazeny jednotlivé příslušné vlastnosti tzv. atributy. V posledním řádku je nastavena třída QoS na nejnižší hodnotu, což je 0 (*Best Effort*).

```
258 OnOffHelper onoffFtp("ns3::TcpSocketFactory", Address(
        InetSocketAddress(remoteAddr, port)));
259 onoffFtp.SetAttribute("OnTime", RandomVariableValue(
        ConstantVariable(1)));
260 onoffFtp.SetAttribute("OffTime", RandomVariableValue(
        ConstantVariable(0)));
261 onoffFtp.SetAttribute("DataRate", DataRateValue(DataRate(131082)));
262 onoffFtp.SetAttribute("PacketSize", UintegerValue(2000000000));
263 onoffFtp.SetAttribute("AccessClass", UintegerValue(UintegerValue(0)
        ));
```

Dále je potřeba definovat aplikace FTP serveru na zdrojovém uzlu a nastavit jeho začátek a konec běhu.

```
266 ApplicationContainer FtpServer = onoffFtp.Install(ftpSource);
267 FtpServer.Start(Seconds(5.0));
268 FtpServer.Stop(Seconds(25.0));
```

Následující řádky zajistí přiřazení cílové IP adresy a portu ke klientské aplikaci. Dále je definována aplikace na klienta a nastaven jeho začátek a konec.

```
270 PacketSinkHelper paketSinkFtp("ns3::TcpSocketFactory",
      InetSocketAddress(remoteAddr, port));
271 ApplicationContainer FtpClient = paketSinkFtp.Install(ftpSink);
272 FtpClient.Start(Seconds(4.0));
273 FtpClient.Stop(Seconds(28.0));
```

Aplikace_Udp()

Druhá aplikace je VoIP, které slouží v programu na demonstraci vlivu zapnuté a vypnuté QoS. Jeho QoS třída byla nastavena na co nejvyšší tzn. *Voice First* z takového důvodu, aby po zapnutí podpory QoS byl upřednostňován a jeho zpoždění a jitter výrazně klesl.

Tato aplikace má taktéž zdrojový *Uzel 3* a cílový *Uzel 7*. Na zdrojovém uzlu běží aplikace UDP server a na cílovém UDP klient. Doba trvání, začátek a konec těchto aplikací je nastavena stejně jako u FTP.

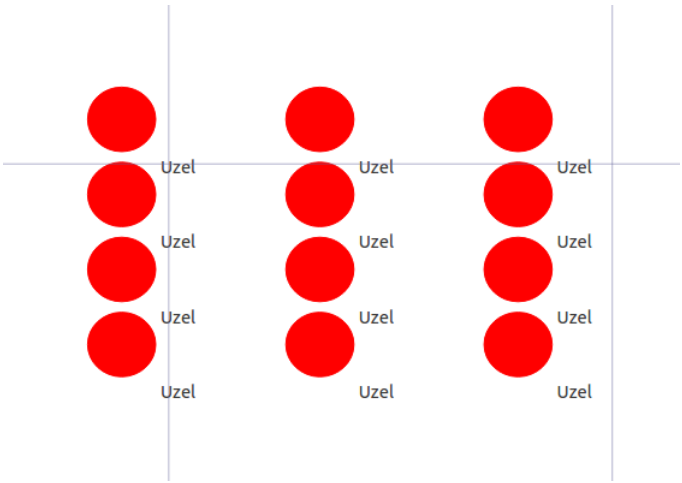
Celá aplikace je taktéž definována pomocí On-Off aplikace a její atributy jsou shodné s aplikacemi FTP, kromě otevření portu (5060) pro komunikaci a zvolení třídy QoS (6).

5.3 Vyhodnocení výsledků

5.3.1 Mobilita uzlů v programu NetAnim

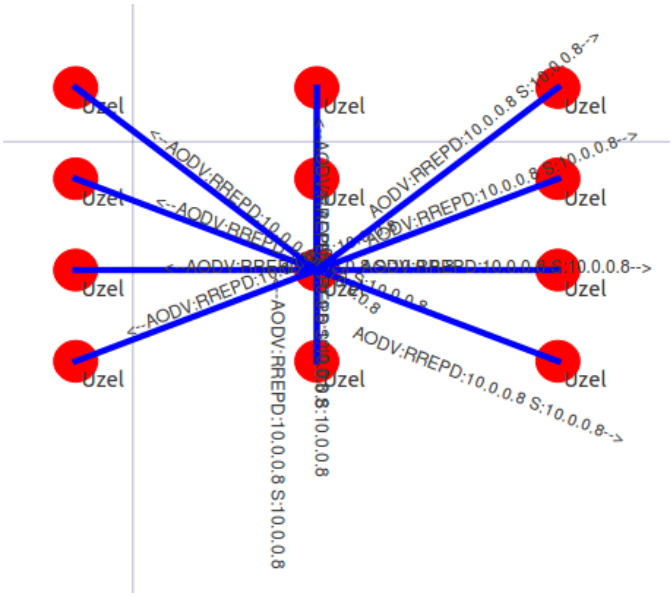
Po otevření souboru, který slouží pro uložení hodnot pro program *NetAnim*, se načítají do programu všechny změny, které byly zaznamenávány. Exportovaný soubor `aodvPotfay_NetAnim.xml` obsahuje následující hodnoty:

- počáteční hodnoty uzlů



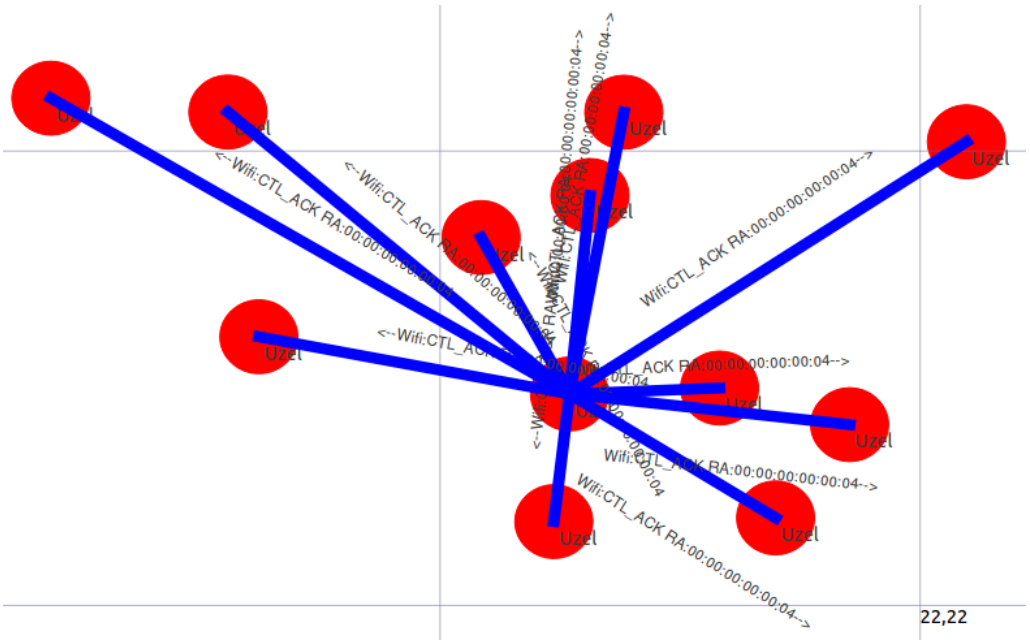
Obr. 5.2: MANET síť v programu *NetAnim*

- informace o paketech (jména, směr paketů atd.)



Obr. 5.3: Ukázka vysílání AODV paketů

- změny polohy uzlů a pohyby uzlů



Obr. 5.4: Pozice uzlů za 15 vteřin

5.3.2 Analýza paketů v programu *WireShark*

Tato podkapitola se zabývá popsáním výsledků, týkající se obsahu paketů, které byly zaznamenány v `pcap` souborech.

Ping

Na kontrolu konektivity a funkčnosti sítě byla spuštěna aplikace *Ping*. Po otevření `pcap` souboru v programu *WireShark* se načítají automaticky všechny pakety, které soubor obsahuje. Po filtraci dle protokolu ICMP byly získány všechny zprávy *Ping*. Ukázka paketu požadavku a odpověď:

```
84 0.082737    10.0.0.8      10.0.0.4      ICMP    122 Echo (ping) request  id=0x0000, seq=0/0, ttl=64
86 0.085457    10.0.0.4      10.0.0.8      ICMP    122 Echo (ping) reply    id=0x0000, seq=0/0, ttl=64
```

Obr. 5.5: Zprávy Ping

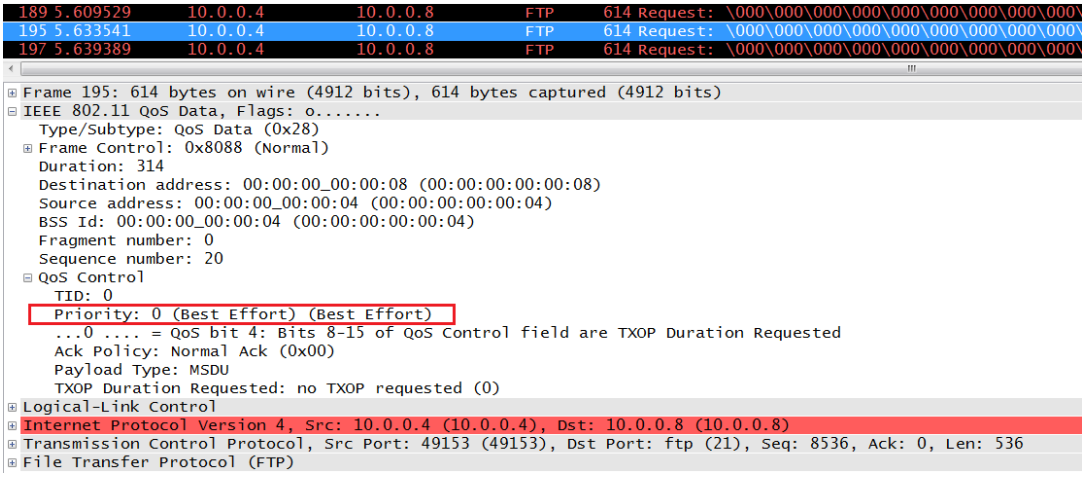
QoS třídy paketů FTP a VoIP

Po otevření `pcap` souboru uzlu 3 nebo 7, byly analyzovány všechny pakety týkající se provozu FTP a VoIP. Po filtrace paketů dle protokolu FTP, byl vybrán jeden paket

a vyhledána QoS data. V označeném řádku můžeme vidět třídu 0 a její označení jako *Best Effort*, což je správně dle nastavení v programu.

Kromě těchto informaci můžeme najít ještě spoustu informací, které byly také nastaveny v simulačním programu, např:

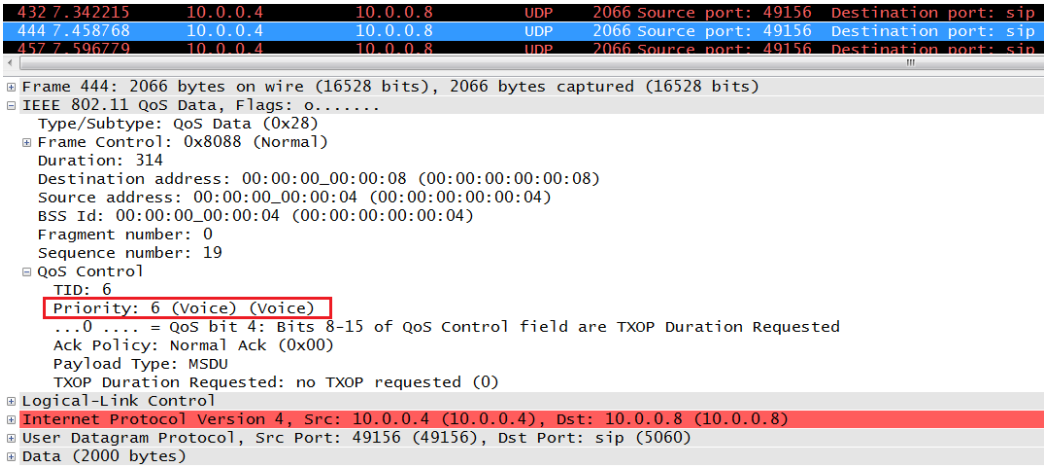
- cílový port 21, přes který celý provoz probíhá
- protokol TCP, který aplikace FTP používá na přenos dat



Obr. 5.6: Ukázka nastavené QoS třídy 0 (*Best Effort*)

Na dalším obrázku jsou pakety VoIP z programu *WireShark*. I tady byla označena priorita, která je třídy 6 (*Voice*) u VoIP provozu. To znamená, že pakety s vyšší prioritou budou upřednostňovány např. při přeposílání paketů. Další informace jsou:

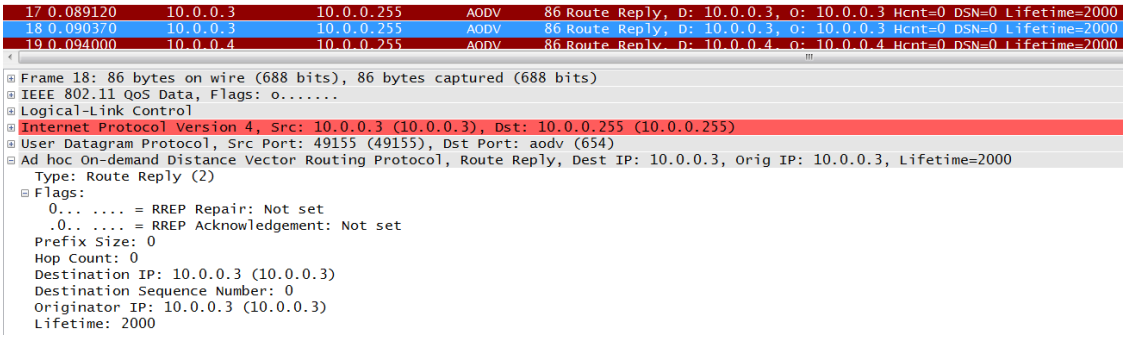
- cílový port 5060
- protokol UDP, který VoIP používá na přenášení dat



Obr. 5.7: Ukázka nastavené QoS třídy 6 (*Voice*)

AODV

Filtrace mezi pakety dle protokolu AODV způsobí vypsání všech paketů, které uzel vysílal v době simulace z důvodu směrování. Funkčnost směrovacího protokolu demonstrují pakety AODV, které jsou pro ukázku:



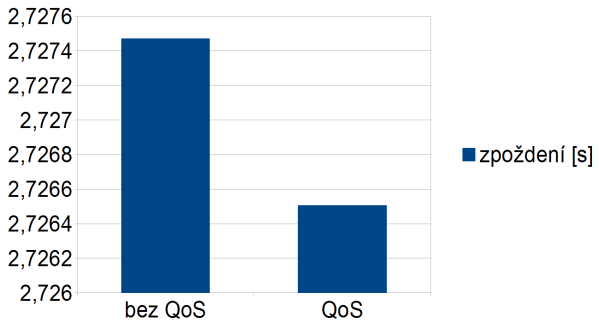
Obr. 5.8: Ukázka AODV paketů

5.3.3 Funkčnost QoS

Tato podkapitola se zabývá vyhodnocením získané hodnoty pomocí *Flow Monitor*. Tyto hodnoty byly:

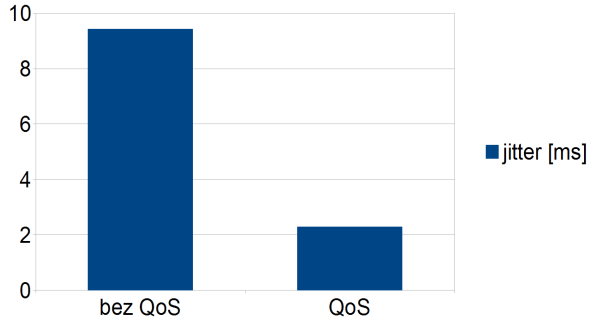
- **`delaySum`** – sečtené hodnoty všech zpoždění v síti, týkající se vybraného provozu, v tomto případě VoIP.
- **`jitterSum`** – podobně sečtené hodnoty všech jitteru v síti, týkající se vybraného provozu VoIP.

V první řadě byla provedena simulace na síti s vypnutou podporou kvality služeb a byly uloženy zmíněné hodnoty. Poté byla opakována simulace se zapnutou podporou kvality služeb a opět uloženy hodnoty. Z těchto hodnot byly sestaveny následující diagramy:



Obr. 5.9: Sumarizované zpoždění VoIP provozu

Po zapnutí kvality služeb byl provoz VoIP upřednostňován, proto nastal pokles v celkovém zpoždění i jitteru, který síť vygenerovala. Za relativně malého poklesu zpoždění po zapnutí QoS, může sestavení cesty na požádání (*on-demand*), což je vlastností směrovacího protokolu AODV. Takže hlídání a sestavení cesty zabírá procentuálně z celého zpoždění velkou část a to skoro před každým vysíláním.



Obr. 5.10: Sumarizovaný jitter VoIP provozu

6 NASAZENÍ AODV PROTOKOLU DO REÁLNÉHO ZAŘÍZENÍ

Pro realizaci implementace směrovacího protokolu AODV byly vybrány typy AODV-UU a WinAODV.

6.1 AODV-UU

Tento typ implementace byl vybrán z důvodu, že implementace směrovacího protokolu AODV do síťového simulátoru `ns-2` a `ns-3` je řešena na základě AODV-UU.

AODV-UU běží na kernelech 2.4 a 2.6, proto byl pro základ implementací vybrán operační systém *Ubuntu 10.04.4 LTS (Lucid Lynx)*.[15, 16] Na tomto typu distribuce běží kernel typu 2.6.32, což vyhovuje pro implementaci.

Operační systém byl stažen ze stránky http://http://releases.ubuntu.com/lucid/. Na stránce ze seznamu obrazů byl vybrán a stažen soubor se jménem *ubuntu-10.04.4-desktop-amd64.iso*. Poté z tohoto obrazu byl vytvořen bootovací CD a USB klíč a operační systém byl nainstalován na jednotlivé přenositelné počítače (notebooky).

6.1.1 Instalace a konfigurace

Po úspěšném nainstalování a spuštění operačního systému *Linux Ubuntu 10.04* je třeba se připojit na internet a stáhnout balíček AODV-UU např. do složky */home/{username}/*. Stahování se provede na stránce http://sourceforge.net/ projects/aodvuu/?source=dlp.[15, 16]

Po stažení se spustí terminál (např. pomocí ikony *Terminal* z menu *Applications* $\longmapsto$ *Accessories* nebo pomocí klávesové zkratky *Ctrl+Alt+T*). Následujícím příkazem se přepneme do administrátorského (v Linuxu to je tzv. *super uživatel*) módu, aby nebylo nutné zadávat `sudo` před každým příkazem.

```
$ sudo -i
```

Instalace

Nejprve je třeba rozbalit stažený balíček. Přepneme se do adresáře, kde byl uložený:

```
$ cd \home\user
```

Balíček se jmenuje *aodv-uu-0.9.6.tar.gz* a rozbalení se provádí pomocí příkazu `tar` s následujícími parametry:

- z – zpracovává soubory, které byly komprimované pomocí gzip.
- x – parametr pro spouštění rozbalení
- v – vypisuje do terminálu, jaké soubory byly rozbalené
- f – přinutí přepsání, pokud již byl extrahován.

```
$ tar -zxvf aodv-uu-0.9.6.tar.gz
```

Po rozbalení systému se vytvoří adresář se jménem `aodv-uu-0.9.6`. Vstoupíme do něj pomocí následujícího příkazu a po té začneme instalaci:

```
$ cd \aodv-uu-0.9.6
```

Pro zbudování (kompilace) programu se v Linuxu používá příkaz *make*. To se provádí pomocí tzv. *Makefile* souboru. Po spuštění tohoto příkazu se vybuduje binární, spustitelný program ze zdrojového kódu.[16]

```
$ make
```

Další krok je instalace programu, který vytvoří AODV démona (aodvd) a modul jádra (kaodv.o) To se provádí následujícím příkazem:[16]

```
$ make install
```

Konfigurace

Konfigurace začíná krokem, kdy je třeba zastavit běh sítového manažera, aby automaticky nezjistil a nepřipojil se k žádné bezdrátové síti.[16]

```
$ stop network-manager
```

Příkaz *iwconfig* se zabývá pouze bezdrátovými prvky bezdrátové karty (jako např.: ESSID, WEP klíče, atd.). Všechny běžné atributy síťové karty (jako např.: IP adresa, maska podsítě atd.) jsou přiřazeny prostřednictvím *ifconfig*.

Následující příkazy slouží pro nastavení jednotlivých parametrů bezdrátové karty. Je třeba nastavit, aby karta fungovala v ad-hoc módu, přiřadíme kanál 11 a staticky nastavíme IP adresu a masku. Kartu je třeba vypnout před nastavením ad-hoc módu, ale s nastavením IP adresy se automaticky zapne.[16]

```
$ ifconfig wlan0 down
$ iwconfig wlan0 mode ad-hoc
$ iwconfig wlan0 essid AODVnetwork
```

```
$ iwconfig wlan0 channel 11
$ ifconfig wlan0 10.0.0.x netmask 255.255.255.0
```

S dalším důležitým příkazem se umožní předávání nebo přesměrování IP paketů z jednoho uzlu na druhý bez jejich lokálního zpracování. Příkaz vlastně dělá to, že napíše do souboru *ip_forward* jedničku (1) a tím aktivuje tuto funkci.[16]

```
$ echo > 1 /proc/sys/net/ipv4/ip_forward
```

Příkaz *modprobe* se používá k načtení kernel modulu. Modul jádra *kaodv* lze nyní spustit zadáním:[16]

```
$ modprobe kaodv
```

Spuštěním AODV probíhá aktivace démona, který se jmenuje jako *aodvd*. Pro jeho použití slouží následující parametry:[16]

- i – pomocí parametru 'i' můžeme použít pro komunikaci jiné rozhraní než výchozí.
- l – spouštění logování debugovacího výstupu do souboru `/var/log/aodvd.log`..
- o – protokol vysílá HELLO zprávy jen v případě přeposílání dat.
- r – spouštění logování směrovací tabulky do souboru `/var/log/aodvd.rtlog`. Periodicky po definovaných N vteřin.
- n – nastaví chování protokolu tak, aby se po přijímání definovaného počtu HELLO zpráv, choval jako soused.
- u – detekce a zabránění proti jednosměrným spojům (experimentální).
- w – povolení podpory Internet brány (experimentální).
- q – nastavení minimální úrovně signálu pro kontrolu paketů.
- V – zobrazit verzi.
- h – vypíše informace o parametrech.

Aktivace AODV démonu (aodvd) se teď spouští s logováním, periodickou aktualizací směrovací tabulky v každých 3 vteřinách a s podporou Internet brány.[16]

```
$ aodvd -l -r 3 -w
```

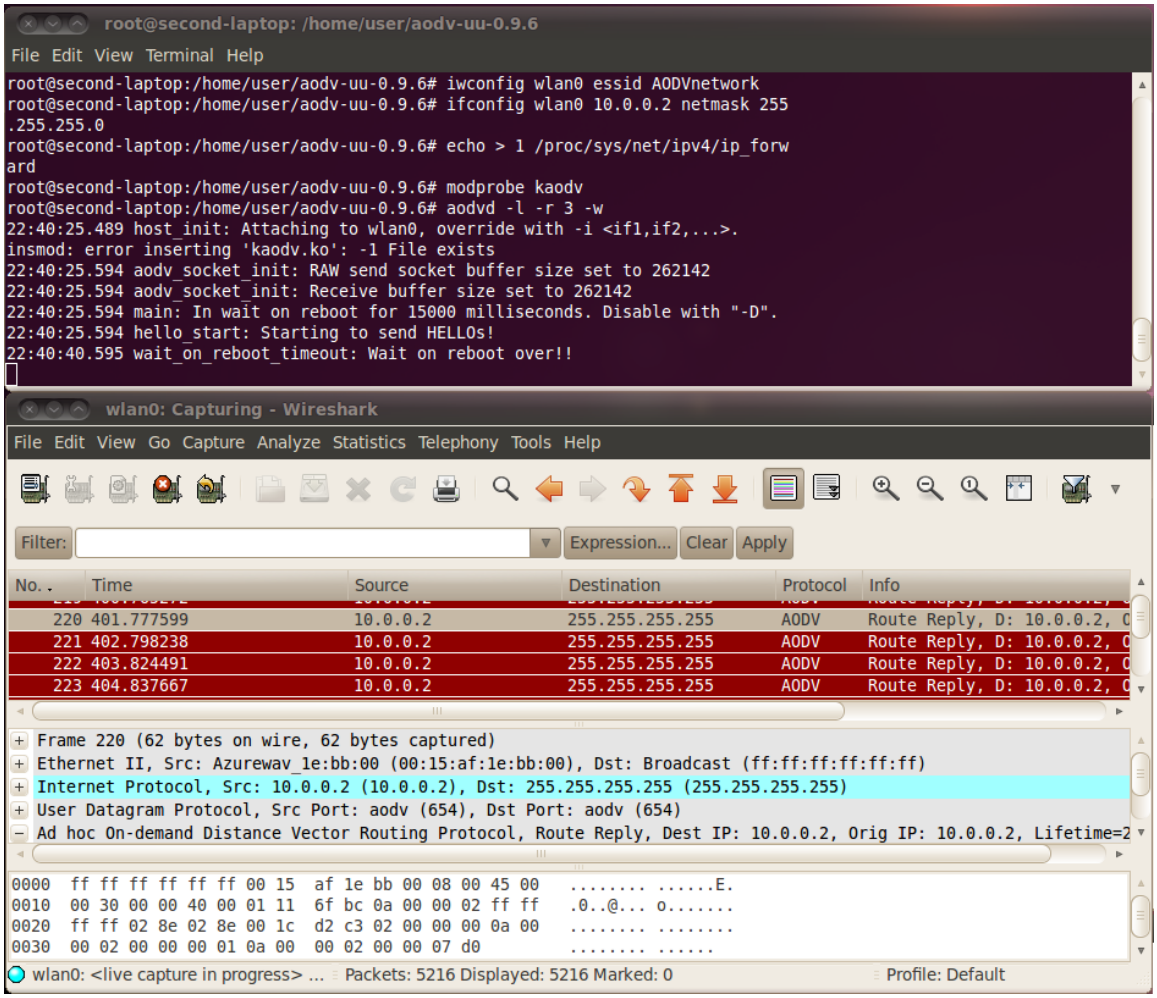
6.1.2 Ověření

Pokud protokol AODV běží, vysílá RREP zprávy, aby našel nějaké sousedy. Funkčnost protokolu lze ověřit pomocí programu, který umí zachytit vysílané pakety. Na to je vhodný program *Wireshark*. Nejjednodušší způsob nainstalování programu *Wireshark* v systému Linux je pomocí příkazu `apt-get install`, ale aby balíček byl přístupný, nejprve musíme aktualizovat seznam balíčků a až poté spustit instalaci:

```
$ sudo apt-get update
$ sudo apt-get install wireshark

$ sudo wireshark
```

Po úspěšné instalaci spouštíme program, kde je třeba vybrat pro sledování rozhraní *wlan0*.

Na následujícím obrázku můžeme vidět, že uzel pravidelně vysílá RREP zprávy.



Obr. 6.1: Vysílané zprávy protokolu AODV

6.1.3 Omezení

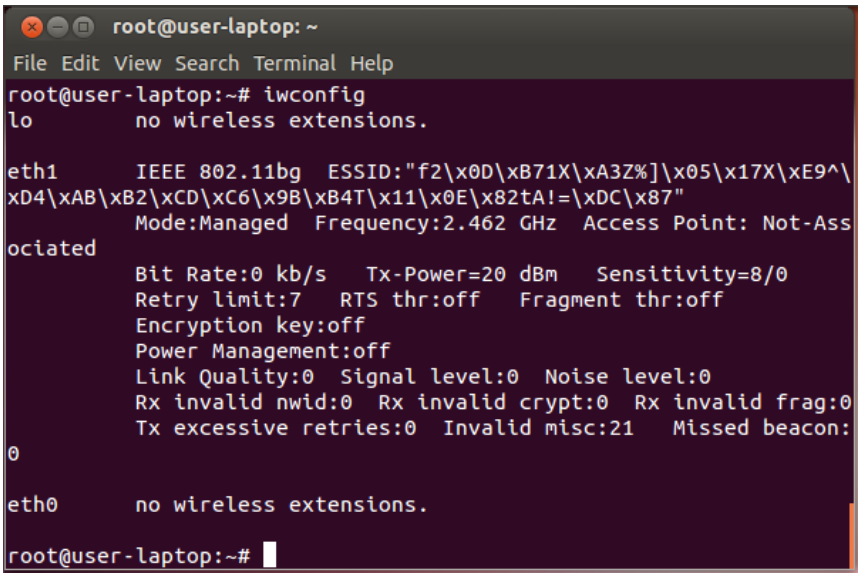
Bohužel existuje velice široké spektrum problémů, které mohou znemožnit realizaci tohoto typu implementací na jednotlivých zařízeních.

Nejnovější počítače

Z těchto důvodu je nejdůležitější to, že AODV-UU běží jen na kernelech typu 2.4 a 2.6. Je takový případ, kde počítač má integrovanou bezdrátovou kartu z nejnovějších sérií, které starší kernely nepodporují a nebo ještě není hotový žádný ovládač. Pokud je nainstalován nejnovější typ distribuce např. Ubuntu 12.10, který má kernel typu 3.5, bezdrátová karta bude fungovat (pokud ovladač existuje), ale AODV-UU při instalaci selže.

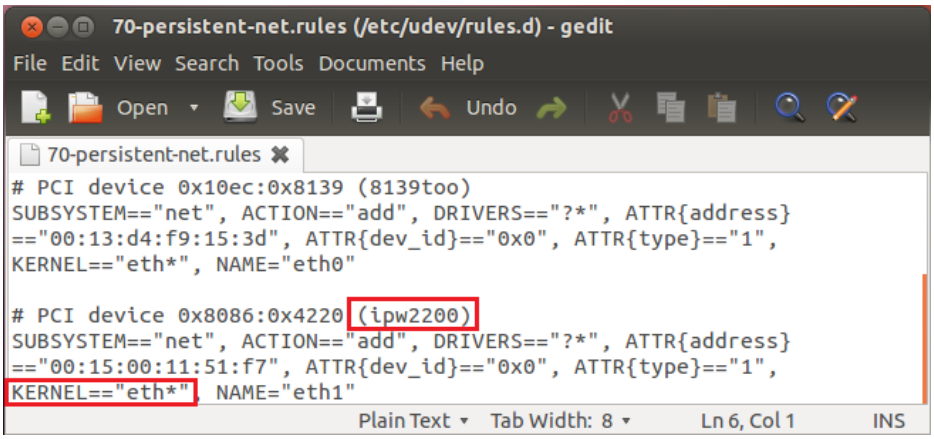
Bezdrátová karta s ethernetovským kernel modulem

Další vážný problém je, že ovladač pro jeden z těch nejrozšířenějších bezdrátových karet není správně vytvořen. Jedná se o bezdrátovou kartu `Intel® PRO/Wireless 2200BG`. Ihned po instalací operačního systému Ubuntu 10.04 rozpozná kartu jako *eth1*. To se ověří pomocí příkazu `iwconfig`:



Obr. 6.2: Výstup příkazu `iwconfig`

Jednotlivé parametry ovladače se ověřují v souboru *70-presistent-net.rules*. Na obrázku 6.3 je výstup tohoto souboru a je označeno, že systém využívá kernel modul ethernetu pro bezdrátové rozhraní. Tento fakt zabrání vysílání AODV zprávy přes daný kanál.

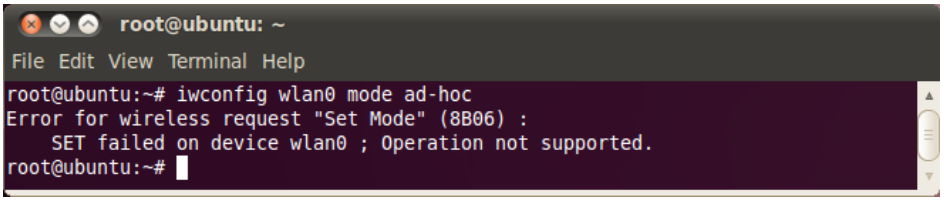


Obr. 6.3: Obsah souboru *70-presistent-net.rules*

Instalace AODV-UU proběhne úspěšně, ale po spuštění démonu *aodvd* se zprávy nevysílají.

Nelze nastavit Ad-hoc režim

Další problém s některým ovládačem je ten, že po vydání příkazu *sudo iwconfig wlan0 mode ad-hoc* se nepřepne do ad-hoc režimu a bez tohoto režimu je nemožné realizovat MANET síť. Tato chybová hláška se objevuje u bezdrátové karty `Realtek RTL8187B`.



Obr. 6.4: Ovladač nepodporuje ad-hoc režim

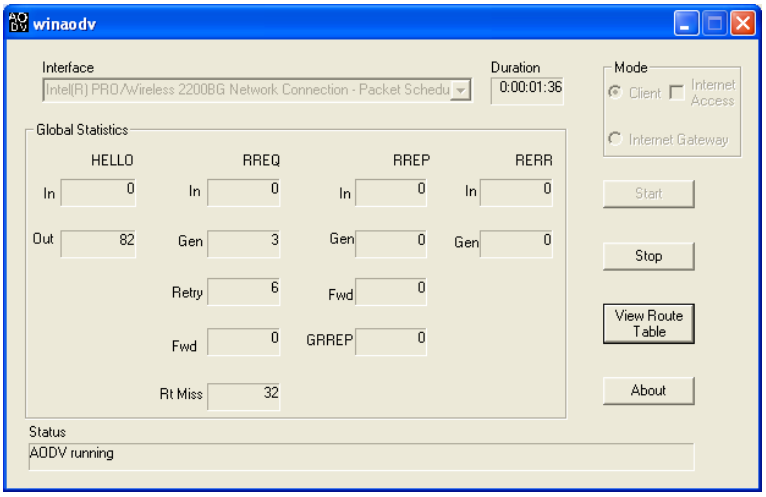
6.2 WinAODV

Tato implementace směrovacího protokolu AODV byla vyvíjena firmou Intel. Byla navržena pro operační systém *Microsoft Windows XP*, ale byla testována i na *Windows 7* a žádné problémy nebyly nalezeny. Implementace byla řešena dle RFC3561. Má dvě části:[17]

- uživatelské rozhraní (*winaodv.exe*)
- přechodný ovladač NDIS – AODV směrování

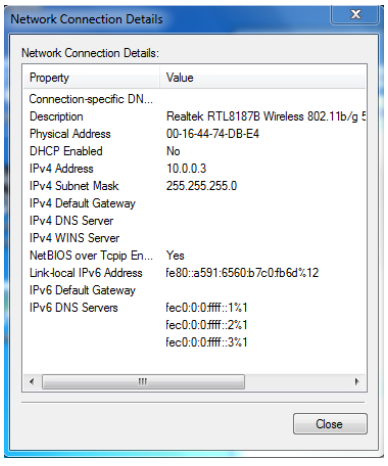
6.2.1 Instalace a nastavení

Projekt WinAODV má domovskou stránku http://moment.cs.ucsb.edu/AODV/aodv-windows.html odkud byla stažena nejnovější verze programu.[17] Po spuštění staženého souboru *windows_aodv_0.1.14.exe* se program nainstaluje běžným způsobem. Aplikace se spouští ikonou na ploše *AODV Routing*. Po spuštění programu se objeví uživatelské rozhraní, jako je na obr.6.5.



Obr. 6.5: Uživatelské rozhraní WinAODV

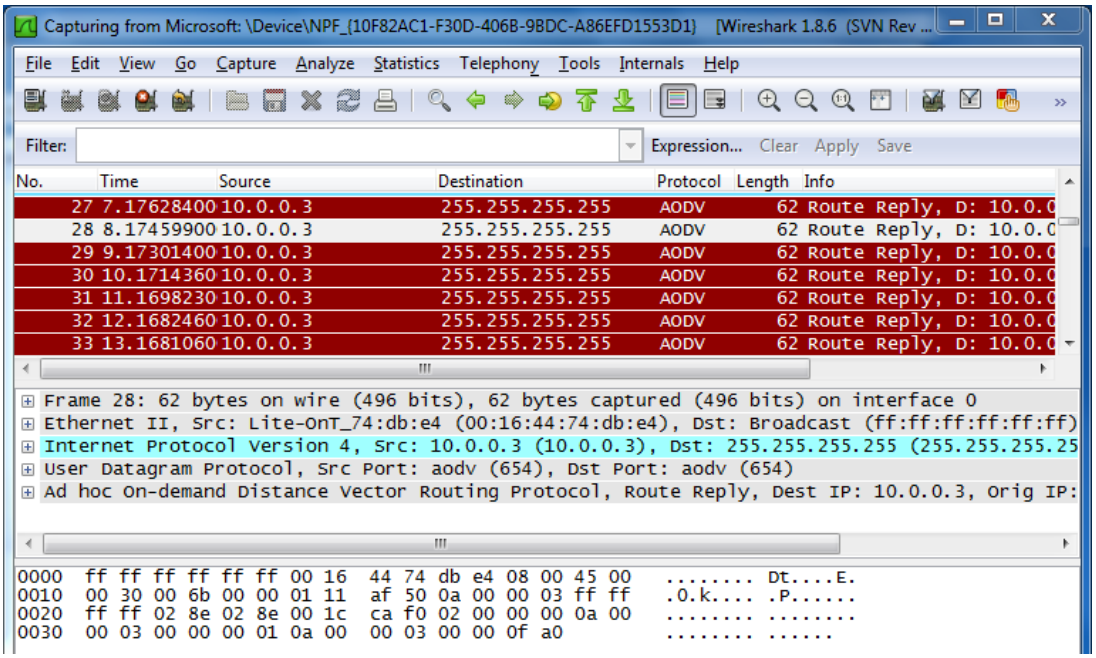
Bylo kliknuto na `Start` a hned poté nastavena IP adresa bezdrátové karty 10.0.0.x a maska 255.255.255.0. Když byla tato akce potvrzena, až potom byla připojena na ad-hoc síť pojmenovanou jako *AODVnetwork*.



Obr. 6.6: Nastavená IP adresa a maska

6.2.2 Ověření

Byl nainstalován program Wireshark pro ověření funkčnosti protokolu AODV.



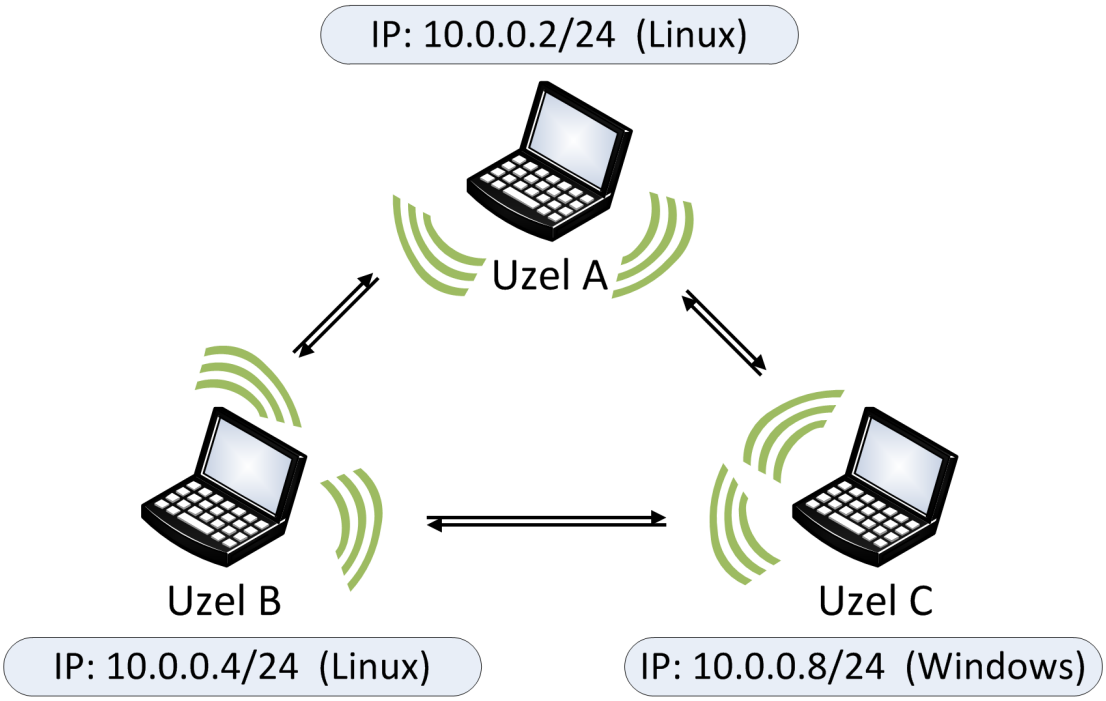
Obr. 6.7: Vysílané zprávy protokolu AODV

7 VYTVOŘENÍ REÁLNÉ SÍTĚ POSTAVENÉ NA SMĚROVACÍM PROTOKOLU AODV

Tato kapitola se bude zabývat vytvořením sítě postavené na směrovacím protokolu typu AODV. Implementace směrovacího protokolu do jednotlivých typů uzlů byla popsána v předchozí 6-té kapitole.

7.1 Topologie

Síť se skládá ze třech uzlů, dva s operačním systémem *Linux* a třetí s *Windows*. Všechny uzly mají dosah na všechny ostatní uzly. Díky směrovacímu protokolu AODV se každý uzel bude snažit chovat jako směrovač, aby se dodržely pokyny MANET sítě. Nezávisle budou kontrolovat totožnost sousedů a vytvářet směrovací tabulky.



Obr. 7.1: Topologie sítě

Popis implementace směrovacího protokolu AODV do uzlů A a B je popsán v kapitole 6.1 a implementace protokolu na platformě Windows (Uzel C) je popsán v části práce 6.2.

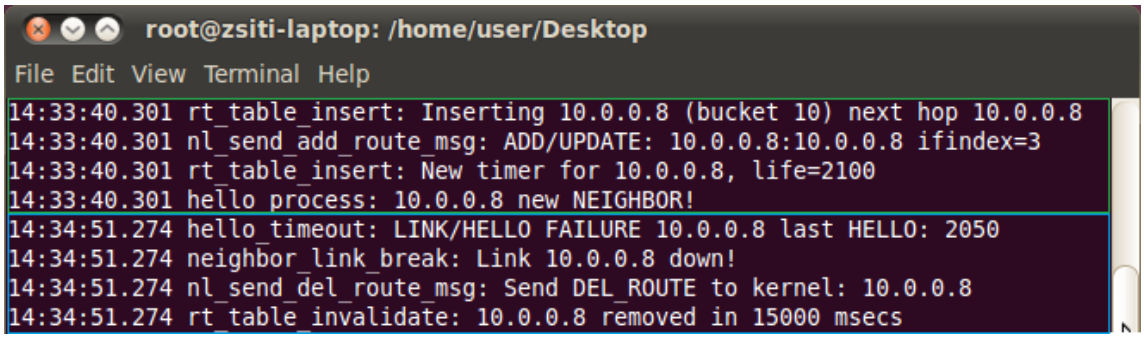
7.2 Dosažené výsledky

Pro dokumentaci funkčnosti sítě byly uloženy směrovací tabulky na všechny uzly, dále logy událostí a pakety.

7.2.1 Uzel A a B

Logy událostí

Soubor, který pravidelně uloží všechny změny o stavu protokolu AODV a o směrovací tabulce, leží v adresáři `\var\log`. Jedná se o soubor se jménem `aodvd.log`. Po spuštění démona AODV (Příkaz na místě 6.1.1) se obsah logu objevuje aktuálně v konzole.



Obr. 7.2: Události na uzlu A: přidání (zelená) a vymazání souseda (modrá)

První řádek označuje nový zápis do směrovací tabulky, uzel s IP adresou 10.0.0.8 (Uzel C). Další řádek ukazuje přidání nového spoje. Třetí ukazuje přiřazení časovače pro nového souseda. Poslední čtvrtý řádek informuje o konečné validaci nového souseda.

V modrém rámci je opačný proces. Vyprší časovač, protože nedostal informaci od souseda C (ztratil konektivitu). Dále označuje linku jako nefunkční a AODV démon vysílá interní příkaz pro vyřazení souseda s IP adresou 10.0.0.8 ze směrovací tabulky. Nakonec probíhá vymazání zápisu z tabulky.

Směrovací tabulky

Směrovací tabulky se taky uloží ve stejném adresáři jako logy událostí `\var\log`. Jméno souboru je `aodvd.rtlog`.

Je vidět, že tabulka na uzlu A obsahuje zápisy uzlů B a C, a uzel B o uzlech A a C. Hodnoty *Destination* ukazují adresy detekovaných sousedů. *Next hop* označuje adresy uzlu, přes kterého může daný uzel komunikovat s jejím vybraným cílem.

HC je zkratka anglického výrazu „Počet skoků“ (*Hop Count*), což slouží k počítání počtů skoků mezi zdrojovým a cílovým uzlem. Jestli je to 1, uzly mají přímý spoj.

```
# Time: 14:36:03.494 IP: 10.0.0.2, seqno: 1 entries/active: 2/2
Destination     Next hop        HC  St. Seqno Expire Flags Iface Precursors
10.0.0.8        10.0.0.8        1   VAL 1     1697         wlan0
10.0.0.4        10.0.0.4        1   VAL 1     1616         wlan0
```

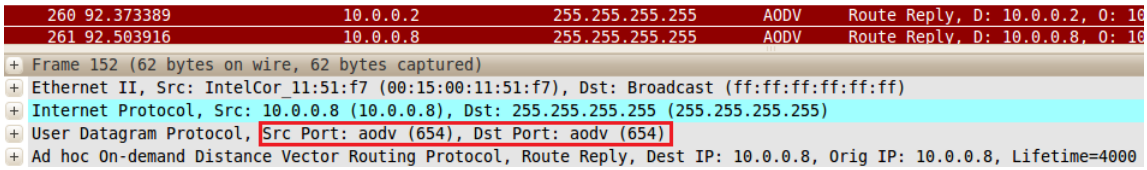
Obr. 7.3: Směrovací tabulka na uzel A

```
# Time: 14:35:34.976 IP: 10.0.0.4, seqno: 1 entries/active: 2/2
Destination     Next hop        HC  St. Seqno Expire Flags Iface Precursors
10.0.0.2        10.0.0.2        1   VAL 1     1550         wlan0
10.0.0.8        10.0.0.8        1   VAL 1     1346         wlan0
```

Obr. 7.4: Směrovací tabulka na uzel B

Kontrola konektivity

Směrovací protokol AODV používá zprávy RREP (*Route Reply*) pro potvrzení sousedství. AODV využívá pro přenos řídících paketů komunikační port 654. Následující obrázek ukazuje totožnost těchto paketů v síti:



Obr. 7.5: Směrovací pakety AODV

Po navázání spojení byla testována konektivita pomocí *Ping* mezi uzly A a B.

123 35.285187	10.0.0.2	10.0.0.4	ICMP	Echo (ping) request
128 36.284091	10.0.0.2	10.0.0.4	ICMP	Echo (ping) request
133 37.285985	10.0.0.2	10.0.0.4	ICMP	Echo (ping) request
138 38.287577	10.0.0.2	10.0.0.4	ICMP	Echo (ping) request
124 35.286892	10.0.0.4	10.0.0.2	ICMP	Echo (ping) reply
129 36.285826	10.0.0.4	10.0.0.2	ICMP	Echo (ping) reply
134 37.287729	10.0.0.4	10.0.0.2	ICMP	Echo (ping) reply
139 38.289310	10.0.0.4	10.0.0.2	ICMP	Echo (ping) reply

Obr. 7.6: ICMP pakety (Ping)

7.2.2 Uzel C

Logy událostí

Na platformě Windows směrovací protokol AODV taky používá log soubor pro uložení událostí. Ten se nachází v domovském adresáři AODV `C:\Program Files\AODV Routing` a soubor se jmenuje jako `aodvlogfile0.txt`.

Následující části logu byly generovány z důvodu nalezení uzlů A a B. Noví sousedé byli zapsáni do směrovací tabulky s potřebnými informacemi, jako cílová adresa (*Destination*), adresa dalšího skoku (*Nexthop*), počet skoků (*Hop*) atd.

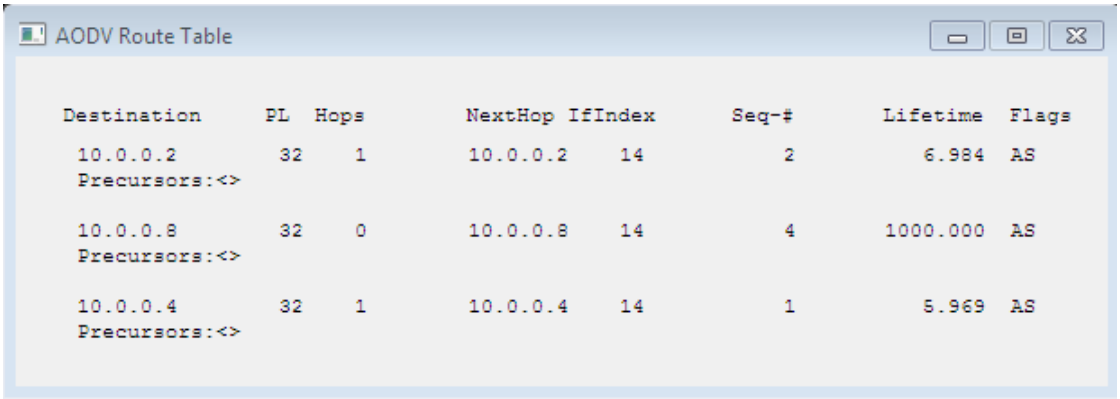
```
Thu Apr 18 14:46:02 2013
Adding 10.0.0.2 to LF All-neighbors Table

Thu Apr 18 14:46:02 2013
Adding 10.0.0.2 to Neighbor Table

Thu Apr 18 14:46:02 2013
Adding Destination:10.0.0.2 Nexthop:10.0.0.2 DSN:1 Hops:1 Lifetime
   :1366289169.234 to Route Table

Thu Apr 18 14:46:02 2013
Adding kernel route Dst:10.0.0.2Mask:0xffffffff Nexthop:10.0.0.2
   Metric:1
```

```
Thu Apr 18 14:46:02 2013
Adding 10.0.0.4 to LF All-neighbors Table

Thu Apr 18 14:46:02 2013
Adding 10.0.0.4 to Neighbor Table

Thu Apr 18 14:46:02 2013
Adding Destination:10.0.0.4 Nexthop:10.0.0.4 DSN:1 Hops:1 Lifetime
   :1366289169.218 to Route Table

Thu Apr 18 14:46:02 2013
Adding kernel route Dst:10.0.0.4Mask:0xffffffff Nexthop:10.0.0.4
   Metric:1
```

Směrovací tabulka

Směrovací tabulka se objeví kliknutím na tlačítko „*View Route Table*“ v uživatelském rozhraní (Ukázka je na obrázku 6.5).

Objeví se směrovací tabulka s informacemi od sousedů. Odlišnost WinAODV od AODV-UU je, že ve směrovací tabulce se objevuje i záznam o daném uzlu, na kterém byla tabulka volána. Tento záznam má hodnotu počet skoků (*Hops*) nulovou, protože se jedná o stejný uzel.



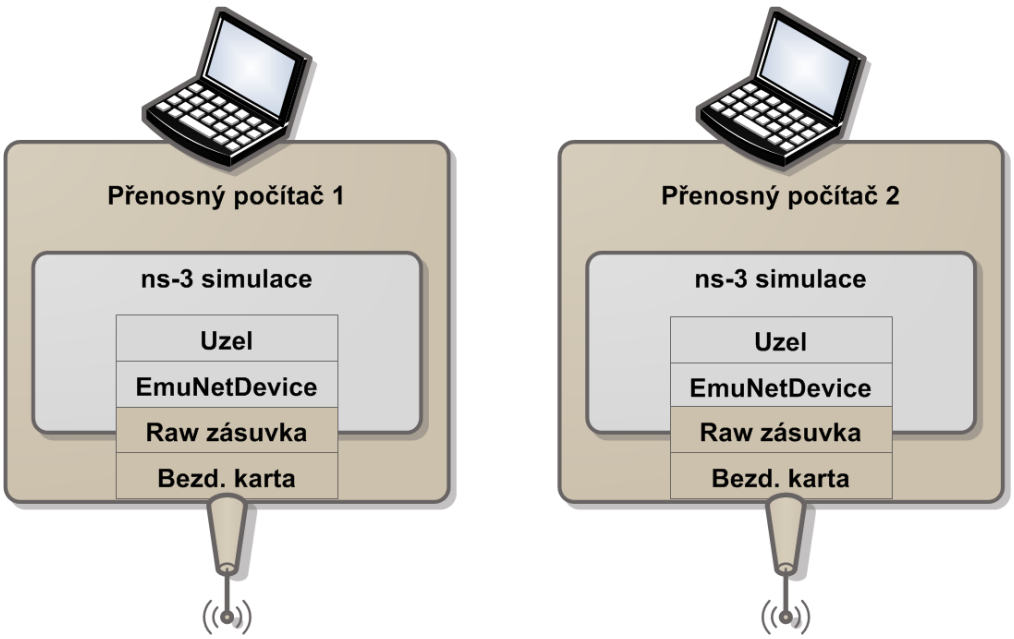
Obr. 7.7: Směrovací tabulka na uzlu C

8 ZAVEDENÍ REÁLNÝCH ZAŘÍZENÍ DO PROCESU SIMULACE

Z důvodu testování vlastnosti sítě, hlavně spojů mezi bezdrátovými uzly, byla vybrána metoda *EmuNetDevice* pro zavedení reálných uzlů do procesu simulace.

8.1 Struktura řešení

Pro realizaci úkolu zavedení reálných zařízení do procesu simulace `ns-3` a simulovat AODV síť, bylo třeba spojit výše popsanou implementaci AODV v kapitole 6.1 se simulátorem `ns-3`. Konkrétní metoda pro zavedení reálných zařízení do simulace byla vybrána metoda *EmuNetDevice* (podkapitola 3.4.1). Byl nainstalován stejný operační systém (Ubuntu 10.04) a postup instalace směrovacího protokolu AODV do přenosných počítačů. Struktura emulace je na následujícím obrázku:



Obr. 8.1: Struktura emulace

Hlavní myšlenka z tohoto řešení je, že na obou počítačích nezávisle běží dva různé simulační *ns-3* skripty. Pakety, které simulace vygeneruje, jsou přenášeny přes bezdrátový spoj mezi počítači. Komunikaci zaručí MANET směrovací protokol AODV.

Pro kontrolu funkčnosti spoje byla vytvořena aplikace *Ping* na obou uzlech, aby kontrolovali konektivitu vzájemně. Byl ještě vytvořen vztah server-klient, kde klient posílá pro server data pomocí protokolu UDP.

8.2 Instalace potřebných programů a balíčků

Po úspěšné instalaci operačního systému linux a implementaci protokolu AODV je třeba nainstalovat podpůrné programy a balíčky.

Prvním krokem je spustit terminál a je třeba aktualizovat databáze dostupných balíčků příkazem:[14]

```
$ sudo apt-get update
$ apt-get install
```

Následující příkazy slouží k instalaci jednotlivých programů, které simulátor `ns-3` vyžívá, `pkg-config` pomocní nástroje pro kompilaci aplikací a `libxml2` slouží pro parsování XML souborů.[14]

```
$ apt-get install pkg-config libxml2 libxml2-dev
```

Balíčky `gcc` a `g++` jsou kompilátory. `Make` je nástroj pro vytvoření spustitelných programů ze zdrojového kódu. Automake vytvoří přenositelné *make* soubory a `autoconf` slouží pro konfiguraci skriptů pro kompilování. Balíček `binutils` je sada programovacích nástrojů pro tvorbu a správu binárních programů.[14]

```
$ apt-get install gcc g++ make automake autoconf binutils
```

`Openssh` je sada počítačových programů poskytujících zašifrované komunikační relace v síti pomocí protokolu SSH. `Openssl` implementuje kryptografické funkce a nabízí pomocné funkce.[14]

```
$ apt-get install openssh-server openssh-client openssl libssl-dev
```

Následující příkaz slouží na instalaci `python-dev`, což obsahuje hlavičkové soubory, statické knihovny a vývojové nástroje pro budování *Python* modulu. `Mercurial` je nástroj na správu zdrojových kódů.[14]

```
$ apt-get install python-dev mercurial
```

`Tcpdump` je program podobný jako *Wireshark* (analyzátor a zachytávač paketů), ale používá se v terminálu. `Gdb` je program pro debugování, což umožní najít chybu v našem kódu. `Emacs` je textový editor.

Příkazem `exit` se odhlásíme z uživatele root-a.[14]

```
$ apt-get install tcpdump gdb emacs
$ exit
```

8.3 Instalace simulátoru ns-3

Instalaci síťového simulátoru `ns-3` je třeba provést v uživatelském adresáři:

```
$ cd /home/user/
```

Vytvoří se adresář `repos` pro repozitáře a příkazem `hg clone` se udělá identická kopie stávajícího repozitáře v síti.[14]

```
$ mkdir repos
$ cd repos
$ hg clone http://code.nsnam.org/ns-3-dev
```

Vytvořil se adresář `ns-3-dev`. Příkaz `./waf` je nástroj určený pro automatické sestavování a instalaci počítačového programu. To se používá pro instalaci a konfiguraci simulátoru `ns-3`.[14]

```
$ cd ns-3-dev
$ ./waf configure -d debug
$ ./waf
```

Pokud vše proběhne úspěšně, simulátor `ns-3` byl nainstalován a je připraven na použití.

8.4 Nastavení bezdrátové karty

Je třeba ještě nastavit bezdrátové rozhraní, tedy jak se bude zařízení chovat v síti. Otevře se pomocí libovolného textového editoru následující soubor s příkazem:[14]

```
$ emacs /etc/network/interfaces
```

Do souboru byly přidány následující dva řádky před podobnou deklaraci primárního síťového rozhraní.

```
auto wlan0
iface wlan0 inet dhcp
```

Řádek začínající slovem auto se používá k identifikaci fyzického rozhraní tzn. `wlan0` bezdrátového rozhraní. Parametr `iface` značí fyzický název rozhraní. Názvu rozhraní následuje název adresní rodiny (výběr adresní verze), která rozhraní používá. To je `inet` pro protokol TCP/IP s adresami IP verze 4. Poslední parametr `dhcp` značí metodu pro získání adresy.

Poslední nastavení je přepnutí bezdrátové síťové karty do promiskuitního režimu. To probíhá následujícím způsobem:[14]

```
$ ifconfig wlan0 promisc
```

Ve výchozím nastavení, když síťové karty obdrží paket, zkontrolují, zda paket patří k sobě. Pokud tomu tak není, karta paket shodí, ale v promiskuitním režimu místo toho bude akceptovat všechny pakety, které přes síťové karty tečou.

Superuser oprávnění vyžaduje nastavit rozhraní do promiskuitního režimu. Například většina nástrojů používá tento režim jako analyzátory a zachytávače paketů.

8.5 Popis simulačních skriptů – Server/Klient

Pro vytvoření simulačního skriptu, je třeba vytvořit nový soubor s příponou `.cc`. Po vytvoření je třeba původní skript a co se v něm objevilo po instalaci smazat, aby v adresáři `scratch` byl jen adresář `subdir` a vytvořený skript.

```
$ cd /home/user/repos/ns-3-dev/scratch | nano uzel1_server.cc
```

To stejné je třeba udělat na druhém uzlu, kde bude vytvořen soubor, pojmenován jako např. `uzel2_klient.cc`.

8.5.1 Inicializace

Hned na začátku programu jsou volány jednotlivé hlavičkové soubory pro zajištění funkčnosti kódu. Níže jsou popsány některé specifikace pro danou problematiku.

Následující hlavičkový soubor slouží pro snadnou instalaci a konfiguraci emulační metody *EmuNetDevice*.[9]

```
#include "ns3/emu-helper.h"
```

Další hlavičkový soubor `mac48-address` se používá pro manipulaci s MAC adresou. Číslo 48 značí délku adresy v bitech.[9]

```
#include "ns3/mac48-address.h"
```

Pro vytvoření aplikaci Ping slouží následující hlavičkový soubor.[9]

```
#include "ns3/v4ping-helper.h"
```

8.5.2 Síťové nastavení

Vytvoří se jeden uzel, resp. Bude to ten uzel, na kterém běží simulace. To probíhá pomocí třídy `NodeContainer`.

```
NodeContainer uzel;
uzel.Create (1);
```

Pro aktivaci směrovacího protokolu AODV slouží třída `AodvdHelper`. Protokol AODV je spojen se sadou internetových protokolů a společně jsou nainstalovány na uzel pomocí příkazu `Install`.[9]

```
InternetStackHelper internet;
AodvHelper aodv;
internet.SetRoutingHelper(aodv);
internet.Install (uzel);
```

Po instalaci protokolů je třeba přidat IP adresy. Musíme se ujistit, že adresování v obou těchto aplikacích je konzistentní. Proto přiřadíme unikátní adresy v rámci sítě. V tomto případě server dostane adresu 10.0.0.4 protože přiřazení začíná od 4-ky.

```
Ipv4AddressHelper ipv4;
NS_LOG_INFO ("Přirazení IP adresy.");
ipv4.SetBase ("10.0.0.0", "255.255.255.0", "0.0.0.4");
Ipv4InterfaceContainer i = ipv4.Assign (d);
```

8.5.3 Nastavení emulaci

Aby bylo možné zprovoznit komunikaci s reálným zařízením, musí být simulátor spuštěn pomocí plánovače v reálném čase. To se provede následujícím příkazem:[9]

```
GlobalValue::Bind ("SimulatorImplementationType", StringValue ("ns3
    ::RealtimeSimulatorImpl"));
```

Komunikace bude probíhat přes bezdrátovou kartu, proto byl nastaven první atribut na `wlan0`. Je třeba ještě nastavit typ enkapsulace a instalovat na uzel.

```
EmuHelper emu;
       emu.SetAttribute ("DeviceName", StringValue ("wlan0"));
       emu.SetAttribute ("EncapsulationMode", StringValue ("Dix"));
       NetDeviceContainer d = emu.Install (uzel);
```

Jediná další změna je, že je třeba zajistit, aby se MAC adresy na obou stranách lišily. Z tohoto důvodu jsou nastaveny unikátní MAC adresy.

```
Ptr<NetDevice> netdevice = d.Get (0);
Ptr<EmuNetDevice> emudevice = netdevice->GetObject<EmuNetDevice>();
emudevice->SetAddress
(Mac48Address::ConvertFrom (Mac48Address ("00:00:00:00:00:01")));
```

8.5.4 Aplikace Ping

Pro kontroly konektivity byla vytvořena aplikace ping. To se uskuteční pomocí třídy `V4Ping`. Je nastavena IP adresa uzlu, se kterou je vzájemně kontrolován živý spoj (Adresa 10.0.0.5 je adresa klienta.).

```
Ptr<V4Ping> ping = CreateObject<V4Ping> ();
ns3::Ipv4Address remoteip = "10.0.0.5";
ping->SetAttribute ("Remote", Ipv4AddressValue (remoteip));
```

Další příkazy slouží na instalaci aplikace do uzlu a na spouštění a zastavení aplikací. V tomto případě se aplikace spouští ve 2. vteřině a skončí v 200. vteřině simulace.

```
n.Get(0)->AddApplication(ping);
ping->SetStartTime (Seconds(2.0));
ping->SetStopTime (Seconds(200.0));
```

8.5.5 Aplikace UDP na straně klienta

Byla vytvořena aplikace pro vysílání UDP paketů od klienta pro server. Aplikace pravidelně vysílá pakety po každé vteřině s velikostí 1024 bajtů. To probíhá 450 krát.

```
uint32_t packetSize = 1024;
uint32_t maxPacketCount = 450;
Time interPacketInterval = Seconds (1);
```

Následující příkazy nastaví aplikaci, aby vysílala pakety s cílovou adresou serveru na portu 21, což je port protokolu FTP. Dále jsou k aplikaci přiřazeny výše nastavené parametry.[9]

```
UdpEchoClientHelper client (serverIP, 21);
        client.SetAttribute ("MaxPackets", UintegerValue (
           maxPacketCount));
        client.SetAttribute ("Interval", TimeValue (
           interPacketInterval));
```

```
        client.SetAttribute ("PacketSize", UintegerValue (
            packetSize));
```

V následujících řádcích probíhá instalace a řízení aplikace. Aplikace začne vysílat v první vteřině simulace a zastaví se po uplynutí 500 vteřin.

```
ApplicationContainer apps = client.Install (n.Get (0));
        apps.Start (Seconds (1.0));
        apps.Stop (Seconds (500.0));
```

8.5.6 Aplikace UDP na straně servera

Je třeba nastavit na straně serveru, aby poslouchal na portu 21. Na daný uzel bylo potřeba nainstalovat aplikaci, spustit a zastavit.

```
UdpEchoServerHelper server (21);
ApplicationContainer apps = server.Install (n.Get(0));
       apps.Start (Seconds (1.0));
       apps.Stop (Seconds (600.0));
```

8.5.7 Řízení simulace

Pomocí následujícího příkazu se aktivuje uložení `pcap` souborů do domovského adresáře simulátoru `ns-3`. Soubor obsahuje zachycené pakety.[9] Je možné analyzovat a číst pomocí programu *tcpdump* nebo *Wireshark*.

```
emu.EnablePcapAll ("emu-klient", true);
```

Následující příkazy slouží na spouštění a zastavení simulace.[9]

```
Simulator::Run ();
Simulator::Destroy ();
```

8.6 Spuštění emulace

Po úspěšném vytvoření simulačních skriptů a uložení na vhodném místě se nejprve spustí simulace na straně serveru:[14]

```
$ ./waf --run uzel1_server
```

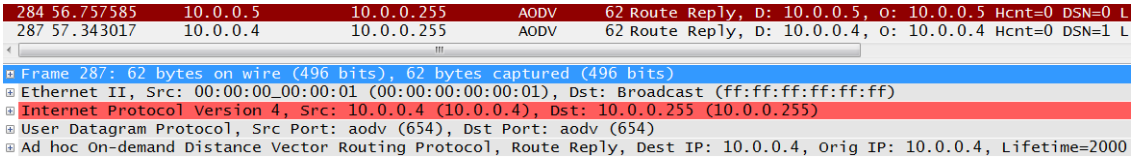
Ihned po té na straně klienta:[14]

```
$ ./waf --run uzel2_klient
```

8.7 Vyhodnocení

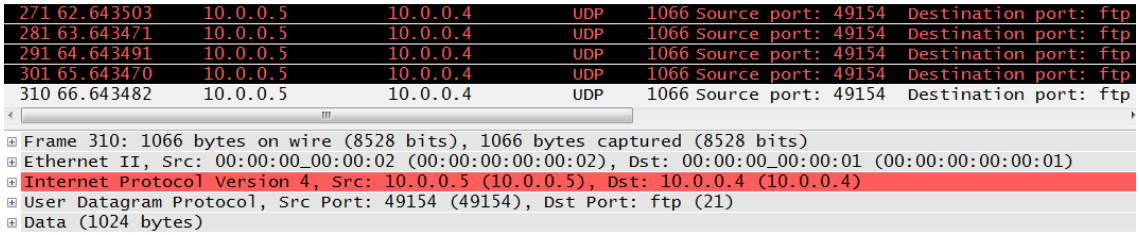
Cílem zavedení reálných zařízení do simulace byl, aby aplikace a tím pádem pakety aplikací byly vygenerovány pomocí simulátoru `ns-3`. Oba uzly se chovají dle vysílaných a přijímaných paketů, jako by se jednalo o reálné počítače. To znázorňuje výstup z programu *Wireshark*.

Analyzované pakety mají plné pole a správné hodnoty. Směrovací protokol navázal spojení pomocí RREP paketů. Stejně se využívá komunikační port 654 pro přenos řídících paketů.



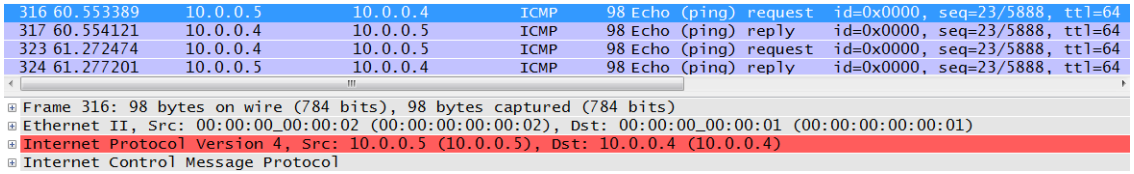
Obr. 8.2: AODV pakety (RREP)

Klient posílá UDP pakety pro server pravidelně po 1 vteřině. To vyplívá i z časů detekování jednotlivých paketů. Dále byly nastaveny MAC adresy uzlů, což je také snadno čitelné z ukázky, že klient má adresu 00:00:00:00:00:02 a server 00:00:00:00:00:01.



Obr. 8.3: UDP pakety (FTP)

Byl napsán kód pro testování konektivity vzájemné mezi uzly (Klient server a opačně). Jako důkaz slouží následující obrázek:



Obr. 8.4: ICMP pakety (Ping)

9 ZÁVĚR

Úkolem teoretické části této diplomové práce bylo seznámit se s MANET (Mobile ad hoc) sítěmi, simulačním prostředím *Network Simulator ns-3*, dále prostudovat a teoreticky zpracovat u těchto sítí stávající možnosti řešení kvality služeb QoS.

V první kapitole bylo řešeno seznámení s vlastnostmi MANET sítí, na co slouží a jak fungují. Byly zpracovány hlavní skupiny protokolů, které se dělí na proaktivní, reaktivní a hybridní. Po dohodě s vedoucím semestrálního projektu byl vybrán protokol ze skupin reaktivních protokolů pro odsimulování v simulačním prostředí `ns-3`. Dále jsou v této kapitole ještě zpracovány základní vlastnosti nejvýznamnějších směrovacích protokolů pro MANET sítě.

V druhé kapitole byly popsány stávající protokoly s podporou kvality služeb. Podrobně byly popsány metriky, které protokoly používají při vybírání cesty. Dále byly zpracovány problémy, které omezují funkčnosti QoS u MANET sítí. První velký problém je právě hlavní vlastnost těchto sítí, tedy že se uzly neustále pohybují. Díky tomu se reaktivní směrovací protokoly stále hlídají a vytvářejí nové cesty k cíli, proto zpoždění narůstá a to je nežádoucí jev. Jak je patrné z výsledků, tento jev byl dokázán pomocí navrženého simulačního programu.

Třetí kapitola zahrnuje teoretické poznatky o simulačním prostředí, v němž simulace celé sítě a směrovacího protokolu byla realizována. Velice užitečná podpora tohoto síťového simulátoru je online dokumentace nazývající se *Doxygen*. V ní je uveden popis všech tříd i ostatní užitečné typy pro využívání všech funkcí simulátoru. Kapitola se dále zabývá dvěmi metodami zapojení reálných zařízení do simulací. Metoda *EmuNetDevice* má charakter pro lepší testování spoje, pro realističtější hodnoty a druhá metoda *TapNetDevice* zapojí reálné uzly do stávající simulace.

Ve čtvrté kapitole jsou zpracovány stávající řešení implementace směrovacího protokolu AODV do reálných zařízení na rozlišných platformách. Byly použity operační systémy Windows a Linux. Z těchto možností implementací byly vybrány *WinAODV* a *AODV-UU* pro splnění dalších úkolů.

Úkolem praktické části bylo vytvořit model MANET sítě, který bude obsahovat podporu směrovacího protokolu AODV. Dále bylo úkolem vyřešit zavedení podpory kvality služeb a zpracovat získané výsledky.

Pátá kapitola práce se zabývá implementací směrovacího protokolu AODV se zavedenou podporou kvality služeb. Zdrojový kód programu byl psán v jazyce C++ a jednotlivé části kódu byly rozčleněny dle jejich funkcí. Takové funkce byly vytvořeny pro vytvoření uzlů, zařízení a aplikací (Ping, FTP, VoIP). Simulační program byl sestaven tak, aby bylo možné sledovat pohyby jednotlivých uzlů a všech paketů, které byly vysílány po celou dobu simulace. Byla sledována i sečtená hodnota zpoždění a jitteru pomocí funkcí *Flow Monitor*.

V podkapitole, která se zabývá vyhodnocením výsledků, byla pomocí programu *NetAnim* demonstrována a popsána stálá změna topologie. Pomocí programu *WireShark* byly odchyceny pakety. Obsah paketů byl vyhodnocen z hlediska úkolů uvedených v zadání, tedy kvality služeb, funkčnosti sítě a směrovacího protokolu. Na důkaz funkční podpory QoS byly v této části práce popsány a demonstrovány ukázkové pakety. Pozitivní výsledky kvality služeb jsou vidět z grafů, které ukazují pokles zpoždění i jitteru po zapnutí podpory QoS. Zdůvodnění malé změny poklesu zpoždění, jak již bylo zmíněno, je kvůli reaktivnímu chování protokolu. Pokles těchto parametrů provozu VoIP dále dokazuje správné rozdělení a upřednostňování paketů s třídou 6(VoIP).

Šestá kapitola práce se zabývá nasazením protokolu AODV do přenosných počítačů. Typ implementace *AODV-UU* v dnešní době nejlépe dokumentována ze všechny možností. Její implementace byla popsána krok-po-kroku. Při implementaci do různých počítačů se narazilo na pár nedostatků, které byly popsány jako omezení tohoto typu implementace.

Dále byla z těchto uzlů vytvořena MANET síť, což je zpracováno v sedmé kapitole. Síť se skládá ze třech uzlů, které komunikují mezi sebou pomocí protokolu AODV. Pro dokumentaci funkčnosti sítě byly uloženy směrovací tabulky na všech uzlech, dále logy událostí a procházející pakety mezi uzly.

Osmá kapitola se zabývá zavedením přenosných počítačů do simulace pomocí metody *EmuNetDevice*. Hlavní myšlenka této metody je, že na každém uzlu běží simulační skript nezávisle na sobě. V tomto řešení byly použity dva uzly, ze kterých jeden je server a druhý klient. Pro demonstrování funkčnosti byly napsány aplikace pro vysílání UDP paketů od klienta pro server a aplikace pro testování konektivity. Dále bylo popsáno krok-po-kroku, jak byl nainstalován simulátor na obou uzlech a další úkoly jako například nastavení bezdrátové karty.

LITERATURA

[1] POTFAY, A. *Simulace směrovacího protokolu OLSR v prostředí OPNET Modeler: bakalářská práce* [online]. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 64 s. Vedoucí práce byl Ing. Pavel Vajsar. Dostupné z URL: <https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=40303>.

[2] *Quality of service in mobile adhoc network* [online]. Dostupné z URL: <http://www.scribd.com/doc/25063350/Qos-in-Manet>.

[3] SANTHI, G., NACHIAPPAN, A. *A Survey Of QoS Routing Protocols For Mobile Ad Hoc Networks* [online]. Department of IT, Department of EEE, Pondicherry Engineering College, India. Dostupné z URL: <http://airccse.org/journal/jcsit/0810ijcsit11.pdf>.

[4] BADIS, H., AGHA, K. *Quality of Service for Ad hoc Optimized Link State Routing Protocol (QOLSR)* [online]. Institut Gaspard-Monge, Marne-la-Vallée, LRI Laboratory, Orsay, France, 2007. Dostupné z URL: <http://www-igm.univ-mlv.fr/~badis/internet-drafts/draft-badis-manet-qolsr-05.txt>.

[5] BADIS, H., AGHA, K. *An Efficient QOLSR Extension Protocol For QoS in Ad hoc Networks* [online]. LRI Laboratory, INRIA Laboratory, France. Dostupné z URL: <http://qolsr.lri.fr/papers/qolsr-flow-vtcfall04.pdf>.

[6] SRIDHAR, S., BASKARAN, R. *A Survey on QoS Based Routing Protocols for MANET* [online]. Department of Computer Applications, S.A.Engineering college Thiruverkadu post, Department of Computer Science and Engineering, CEG, Guindy, Anna University, Chennai, India. Dostupné z URL: <http://www.ijcaonline.org/volume8/number3/pxc3871671.pdf>.

[7] SIVAKUMRA, R., SINHA, P., BHARGHAVAN, V. *CEDAR: A Core-Extraction Distributed Ad Hoc Routing Algorithm* [online]. IEEE journal on selected areas in communications, Vol. 17, No.8, August 1999. Dostupné z URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=779926&userType=&tag=1>.

[8] *ns-3 Manual* [online]. Release ns-3.15, ns-3 project, October 09, 2012. Dostupné z URL: <http://www.nsnam.org/docs/release/3.15/manual/ns-3-manual.pdf>.

[9] *ns-3 Documentation* [online]. Doxygen 1.8.1.2, 2012. Dostupné z URL: <http://www.nsnam.org/docs/release/3.15/doxygen/index.html>.

[10] KULADINITHI, K., UDUGAMA, A., FIKOURAS, N., GÖRG, C. *Experimental Performance Evaluation of AODV Implementations in Static Environments* [online]. ComNets, Universität Bremen, Otto-Hahn-Allee NW1, 28359 Bremen, Germany. Dostupné z URL: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.83.1287&rep=rep1&type=pdf>.

[11] GUPTA, P., TUTEJA, R. *Implementation study of AODV for Microsoft Windows CE platform* [online]. MCA, Manav Rachna College of Engineering, Faridabad, N.C. Institute of Computer Sciences, Israna, Panipat, India. Dostupné z URL: <http://ijcsi.org/papers/IJCSI-9-2-3-235-241.pdf>.

[12] GUPTA, P., TUTEJA, R. *Design Strategies for AODV Implementation in Linux* [online]. MCA, Manav Rachna College of Engineering, Faridabad, N.C. Institute of Computer Sciences, Israna, Panipat, India. Dostupné z URL: <http://www.scribd.com/doc/46132806/Design-Strategies-for-Aodv-Implementation-in-Linux>.

[13] DHARMARAJU, D., KARIR, M., BARAS, J., DAS, S. *An Implementation Study of Multicast Extensions of AODV* [online]. Center for Satellite and Hybrid Communication Networks, Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742, USA. Dostupné z URL: <http://terpconnect.umd.edu/~karir/papers/spects03.pdf>.

[14] *ns-3 Documentation* [online]. ns-3 MediaWiki, 2008. Dostupné z URL: <http://www.nsnam.org/wiki/index.php/HOWTO_use_VMware_to_set_up_virtual_networks_(Windows)>.

[15] *AODV-UU WebSite* [online]. Dostupné z URL: <http://aodvuu.sourceforge.net/>.

[16] *AODV-UU Install* [online]. Dostupné z URL: <http://abdusyarif.wordpress.com/2011/01/18/installing-aodv-uu-step-by-step/>.

[17] *AODV for Microsoft Windows* [online]. Dostupné z URL: <http://moment.cs.ucsb.edu/AODV/aodv-windows.html>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

Ad-hoc – Dočasné spojení mezi rovnocennými prvky.

ANSN – Advertised Neighbor Sequence Number

AODV – Ad-hoc On-demand Distance Vector

AODV-UU – Ad-hoc On-demand Distance Vector - Uppsala University

API – rozhraní pro programování aplikací – Application Programming Interface

ASL – Ad-hoc podporující knihovna – Ad-hoc Support Library

ATM – asynchronní režim přenosu – Asynchronous Transfer Mode

CEDAR – Core Extraction Distributed Ad hoc Routing

CEQMM – Kompletní a účinný QoS model pro MANET sítě – Complete and Efficient Quality of Service Model for MANET

CD – kompaktní disk – Compact Disc

DAG – Directed Acycling Graph

DiffServ – Diferencované služby – Differentiated Services

DSR – Dynamic Source Routing

ESSID – Rozšířený identifikátor služeb – Extended Service Set Identification

FDDI – Fiber Distributed Data Interface

FSM – konečný automat – Finite-State Machine

FQMM – Flexibilní QoS model pro MANET sítě – Flexible QoS Model for MANET

FTP – protokol pro přenos souboru – File Transfer Protocol

GPS – globální polohový systém – Global Positioning System

GRP – Geographic Routing Protocol

HC – Počet skoků – Hop Count

ICMP – Internet Control Message Protocol

IntServ – integrované služby – Integrated Services

IP – internet protokol – Internet Protocol

LTS dlouhodobá podpora – Long term Support

MAC – Media Access Control

MANET – mobilní ad-hoc sítě – Mobile Ad-Hoc Network

MAODV – Multicast Ad-hoc On-demand Distance Vector

MID – deklarace více rozhraní – Multiple Interface Declaration

MPR – Multipoint Relay

NDIS – ovládač síťového rozhraní – Network Driver Interface Specification

ns-3 – simulační prostředí – Network Simulator ns-3

PDA – kapesní počítač – Personal Digital Assistant

OLSR – optimalizován Link State směrovací protokol – Optimized Link State Routing Protocol

QAODV – AODV s podporou QoS – AODV with QoS

QMPR – QoS Multipoint Relay

QOLSR – OLSR s podporou QoS – OLSR with QoS

QoS – kvalita služeb – Quality of Service

RREP – zpráva protokolu AODV, odpověď na nové cesty – Route Reply

SSH – Secure Shell

TC – Topology Control

TCP – Transmission Control Protocol

TORA – Temporally Ordered Routing Algorithm

UCSB AODV – Ad-hoc On-demand Distance Vector – University California, Santa Barbara

UIUC AODV – Ad-hoc On-demand Distance Vector – University of Illinois, Urbana-Champaign

UDP – User Datagram Protocol

USB – Univerzální sériové rozhraní/sběrnice – Universal Serial Bus

VoIP – přenos digitalizovaného hlasu v paketů – Voice over Internet Protocol

WEP – soukromí ekvivalentní drátovým sítím – Wired Equivalent Privacy

Wi-Fi – bezdrátové sítě

WiMAX – Worldwide Interoperability for Microwave Access

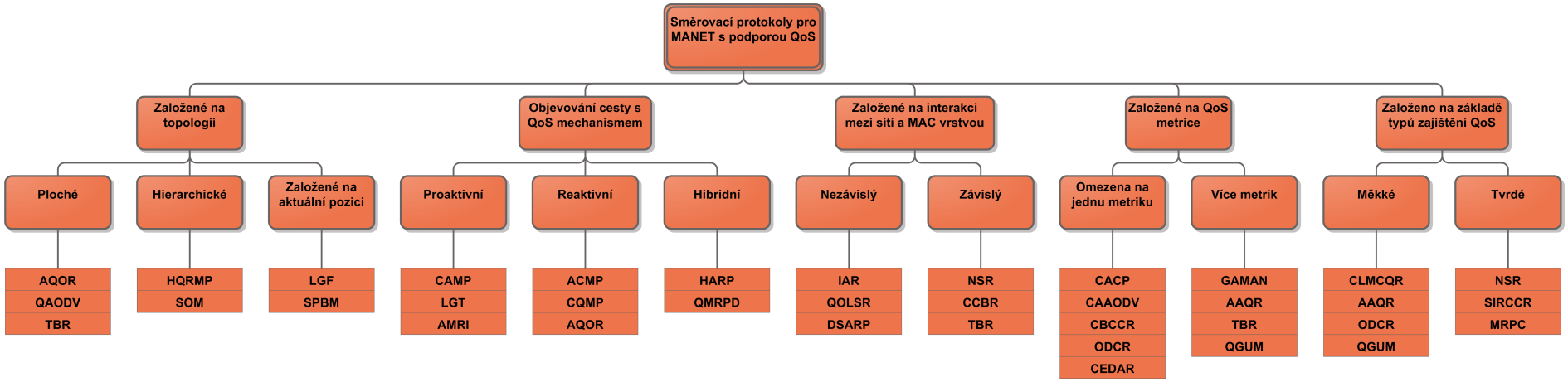
WinAODV – Windows Ad-hoc On-demand Distance Vector

XML – Extensible Markup Language

YANS – Yet Another Network Simulator

SEZNAM PŘÍLOH

A SMĚROVACÍ PROTOKOLY PRO MANET S PODPOROU QOS



B POUŽITÉ PŘENOSNÉ POČÍTAČE

V kapitole 7:

	Typ počítače	**Typ bezdrátové karty**
Uzel A	ASUS F3JR	Intel(R) PRO/Wireless 3945ABG
Uzel B	ASUS X51R	Atheros AR5006EG
Uzel C	ASUS A3500E	Intel(R) PRO/Wireless 2200BG

Tab. B.1

Zkoušené implementace na dalších počítačích:

Typ počítače	**Typ bezdrátové karty**
TOSHIBA Satellite Pro L40	Realtek RTL8187B
SONY Vaio SVE1112M1EP	Qualcomm Atheros AR9485WB-EG

Tab. B.2

C OBSAH PŘILOŽENÉHO CD

1. Elektronická verze vypracované diplomové práce ve formátu `pdf`.
2. Simulační skript se zaměřením na QoS
3. Simulační skripty s metodou `EmuNetDevice` - Server/Klient